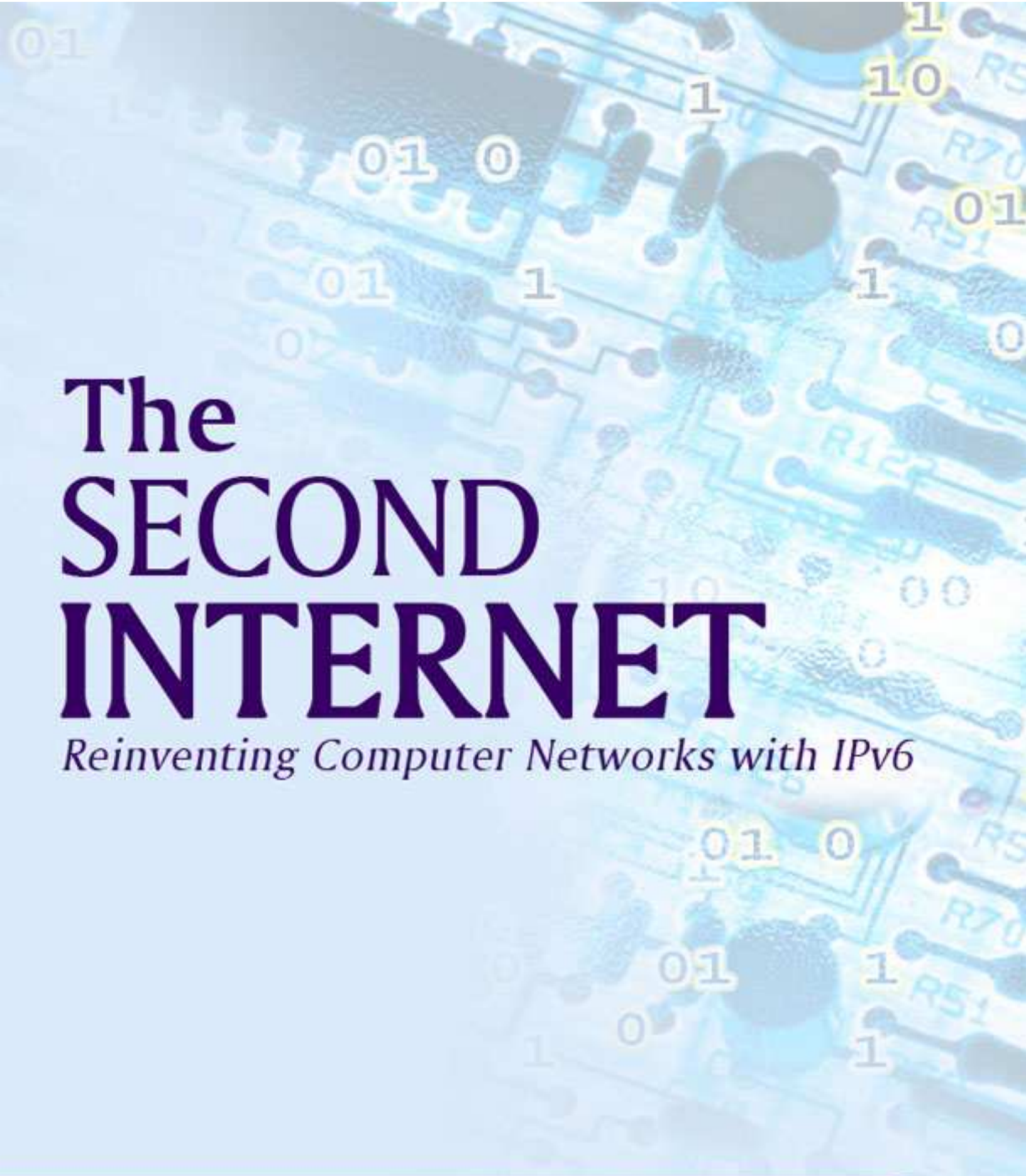


The SECOND INTERNET

Reinventing Computer Networks with IPv6

Lawrence E. Hughes

October 2010

With thanks to
Michael A. Russell
for extensive
proofreading help.

Copyright & License

Publisher: InfoWeapons (www.infoweapons.com)
Unit 801, Skyrise Bldg #3, AsiaTown IT Park, Lahug, Cebu City, Cebu 6000 PHILIPPINES

The author can be contacted at: mailto:lhughes@infoweapons.com or mailto:lhughes@hughesnet.org.
The primary website for this work is *www.secondinternet.org*.

ISBN-10: 098-284-630-4
ISBN-13: 978-0-9828463-0-8

Table of Contents

Foreword

The Chips are Down! However Knowledge is Sadly Missing!

The word “Internet” has become a household name in every language without any translation. Even the French have kept the same name while normally they tend to create a French version for any English name to make it sound like it’s invented in France. Now when you ask normal users how the Internet really functions, be prepared to be surprised by the sparse response and accept their kind apology that they had no time to delve into this complex world, understandably.

When you ask Internet experts on the current Internet Protocol (IPv4), you will be enriched by their prolific and visionary thoughts of what you can do with the Internet and most probably that it can even solve world hunger. However when you ask these same experts about the new Internet Protocol version 6 (IPv6), you will find only a few that can answer with high precision how the new Internet based on IPv6 functions, how it will be installed or how it will enhance the current Internet. I guess most Internet experts have by now understood that the visible difference between IPv4 and IPv6 is the size of the address space moving from a limited to virtually unlimited resource (from 4.3 Billion addresses to 340 Billion Billion Billion Billion addresses).

While basic IPv6 was designed and standardized between 1994 and 1998 and deployment has been happening at a slow pace for the last 10 years, it is astonishing to see the same historical deployment patterns of the current Internet Protocol (IPv4). That was designed between 1972 and 1980 with first deployment in 1981. It had to wait for ten years until 1991 for the Internet to be opened for public use per US Congress agreement. The number of IPv4 experts was quite small and not surprisingly it’s the same level of IPv6 experts that we have now.

The Internet community is asking for killer apps to facilitate justifiable deployment of IPv6. Now, without educated engineers at developer’s level and ISP levels, it is unreasonable to expect creation of such apps that would benefit from the new built-in features in IPv6. The principal feature of IPv6 is the restoration of the end-to-end model on the back of which the Internet was built on in the first place. The e2e model restores e2e connectivity, e2e security, e2e QoS, node reachability, remote access for maintenance and network management purposes. Essential features have been tightly redesigned like mobility, Multicast, auto-configuration, to take the Internet where it has not gone before. IPv6 will take the Internet into commodity services adding networking value to services like RF-ID and sensors. IPv6 will open new paradigms for Internet of Things, Smart Grids, Cloud Computing, Smart Cities, 4G/LTE services, etc.

To realise this we need to have engineers professionally trained with IPv6 eyes and not with IPv4 eyes. A recent survey on IPv6 training and studies at universities has demonstrated that IPv6 training and courses are way too embryonic to have any critical impact. Patching IPv6 with IPv4 thinking would be just extending the IPv6 address space to the Internet and not fully exploiting the rich set of new features still invisible to the normal engineer. Deploying IPv6 without upfront integration of IPv6 security and privacy is re-doing the same mistake done in the deployment of IPv4. This is even defeating the prime purpose of fixing thing like security in the Internet. It is estimated that some 20 million engineers are

working on the current Internet worldwide at ISPs, corporate and all other public and private organisations and they will need training on IPv6. This is a gigantic task since it's the first upgrade of the Internet and most probably the last one for decades to come.

There are also tools that address IPv6 issues that have been designed by the author that will play key roles in the deployment of IPv6 such as DHCPv6, DNSSECv6, IDN, etc. The author brings practical and hands-on experience to take enthusiast engineers to the next level with astonishing intricate knowledge pretty rare to find in this diffuse Internet world not knowing who holds the truth.

I encourage everyone to read this book as you will enjoy it like I did as the author is pretty crystal clear, precise, authoritative and written directly from his heart.

Latif Ladid
President IPv6 Forum
Senior Researcher, University of Luxembourg
Emeritus Trustee, Internet Society
UN Strategy Council member

Chapter 1 – Introduction

1.1 – Why IPv6 is Important

The First Internet (which I now call the *Legacy* Internet) is 27 years old. Think about what kind of CPUs, amount of RAM, and which Operating System you were using in 1983. Probably a Z80 8-bit CPU with 64 Kilobytes of RAM and CPM/80, or if you were a businessman, an 8088 "16-bit" CPU and DOS 1.0. If you were *really* lucky, you might have had an expensive Hard Disk Drive with a massive TEN megabytes of storage. What, many of you reading this weren't even alive then? Ask your father what personal computing was like in 1983. I've been building, programming and applying personal computers since my Altair 8800 in 1975. Hard to realize that is 35 years ago. Since 1983, network speeds have increased from 10 Mbit/sec to 100 Gbit/sec (10,000 fold increase). But we are still using essentially the same Internet Protocol. Think it's about time for an upgrade?

The First Internet has impacted the lives of more than a billion people. It has led to unprecedented advances in computing, communications, collaboration, research and entertainment (not to mention time-wasting and even less savory activities). The Internet is now understood to be highly strategic in every modern country's economy. It is difficult to conceive of a country that could exist without it. Many enormous companies (such as Google) would not have been possible (or even needed) without it. Staggering amounts of wealth have been created (and consumed) by it. It made "snail mail" (paper mail physically delivered) follow the Pony Express into oblivion (amazingly, governments everywhere are trying to keep Post Offices going, even though most lose gigantic amounts of money every year). The number of E-mails sent *daily* is 3 to 4 times the number of first class mails sent *annually* (both in the United States).

Estimates are that there are currently about 1.3 billion nodes (computers, servers or other network devices) connected to the First Internet. Many of those have more than one user (as in Cyber cafes).

1.1.1 – But Wait, There's More….

If you think *that's* impressive, wait until you see what its rapidly approaching successor, the *Second Internet* (made possible by IPv6) will be. Entirely new and far more flexible communication and connectivity paradigms are coming that will make E-mail and texting seem quaint. Major areas of the economy, such as telephony, entertainment, almost *all* consumer electronic devices (MP3 players, TVs, radios) will be heavily impacted, or even collapse into the Second Internet as Yet More Network Applications (like E-mail and web did in the First Internet). The number of connected nodes will likely explode in the next 5-10 years by a factor of a *hundred* or more (not by 100%, I said by a *factor* of 100, which is 10,000%). The First Internet (the one you are using today, based on IPv4) that you think is so pervasive and so *cool*, is less than 1% of the expected size of the Second Internet. One of the popular terms being used to describe it is *pervasive computing*. That means it is going to be *everywhere*.

1.1.2 – Flash! The First Internet is Broken!

Most importantly, in the process of keeping IPv4 around *too long*, we've already *broken* the First Internet badly with something called NAT (Network Address Translation). NAT has turned the Internet into a one-way channel, introduced many really serious security issues and is impeding progress on newer applications like Voice over Internet Protocol (VoIP) and Internet Protocol Television (IPTV). You can easily make outgoing connections to servers like *www.cnn.com*, but it is difficult or impossible for other people to make connections to *you*. It has divided the world into a few *producers* (like cnn.com) and millions of *consumers* (like you). In the Second Internet, anyone can be a *prosumer* (producer *and* consumer). NAT was a necessary evil to keep things going until the Second Internet was ready to be rolled out. NAT has now served its purpose, and like crutches when your broken leg has healed, should be cast aside. Its only purpose was to extend the life of the IPv4 address space while the engineers were getting IPv6 ready.

Using a "horses and cars" metaphor, there is no reason to wait for the last horse to die (the last IPv4 address to be given out) before we start driving cars (deploy IPv6). Good news, everyone! IPv6 is ready for prime time today. My home is already fully migrated to dual stack (IPv4 + IPv6). And that's in the Philippines!

1.1.3 – Wait, How Can the Internet Grow by 100 Fold?

If there are over a billion nodes on the First Internet, and there are just over 6 Billion people alive, how can it *possibly* grow by more than 100 fold? The key here is to understand that the Second Internet (based on IPv6) is *the Internet of Devices*. A human sitting at a keyboard will be a relatively rare thing, although IPv6 will make it far easier and cheaper to bring the *next billion* humans online using IPv6's advanced features and almost unlimited address space. Many Asian countries and companies (who routinely have 5 to 10 year horizons in their planning) already consider IPv6 to be one of the most strategic and important technologies anywhere, and are investing heavily in deploying it. 2010 is the *tipping point* for IPv6. Adoption curves are starting to climb at steep rates reminiscent of the adoption of the World Wide Web back in the early 1990s. By March 2012 (when the last IPv4 address will likely be allocated to some lucky end-user), the migration to IPv6 will be well underway in most leading countries, and *completed* in many Asian countries.

1.1.4 – Why is 2011 a Significant Year for the Second Internet?

There is an entire chapter in this book on the *depletion of the IPv4 address space.* What this means (in English) is that we are *running out* of IP addresses for the First Internet. This will be a *very* important event in the history of the Internet. We nearly ran out in 1997, and only managed to keep the Internet going through some clever tricks (NAT and Private Addresses), kind of like using private extension numbers in a company PBX phone system. However, even with this trick (which is now causing major problems), we are about to run out for good. The folks that create the Internet *don't have any more clever tricks up their sleeves*. All the groups that oversee the Internet, like the Internet Assigned Numbers Authority (IANA), the Internet Corporation for Assigned Names and Numbers (ICANN), the Internet Society (ISOC), the Internet Engineering Task Force (IETF) and the Regional Internet Registries (RIRs) have been saying for some time that the world *has* to migrate *now*. The five Regional Internet Registries are ARIN, RIPE NCC, APNIC, LACNIC and AfriNIC. They should know. They are the ones that

give out IP addresses. They know that the barrel is almost empty. We've *got* to increase the number of globally unique Internet addresses, which has some far reaching consequences.

1.2 – An Analogy: the Amazing Growing Telephone Number

When I was very young, my family's telephone had a 5 digit phone number (let's say it was 5-9999). As the number of phones (and hence unique phone numbers within my geographic region) grew, the telephone company had to increase the length of everyone's phone number. Our number became 385-9999. This was enough to give everyone in my area a unique number, and we could ask the nice long distance operator to connect us to people in other areas when we wanted to talk with them. When the telcos introduced the miracle of *Direct Distance Dialing*, our phone number grew to 10 digits: (904) 385-9999. In theory, this could provide unique numbers to 10^{10} (10 billion) customers. In practice some digit patterns cannot be used, so it is somewhat less than that, and today many people have multiple phone numbers (landline, cell phone, fax, modem, VoIP, etc.). Estimates are that the current supply of 10 digit numbers will last U.S. subscribers at least 50 more years. Increases in the length of phone numbers may be an inconvenience to end users (and publishers of phone books), but the tricky problems are mostly in the big phone switches. Phone number lengths have been changed several times without leading to the collapse of civilization.

One popular estimate (from NetCore) is that the IP addresses for the First Internet will be *all gone*, *history*, *used up* by September 16, 2011 (as estimated on February 15, 2010, subject to many revisions before that last address is assigned, but probably to *earlier* dates, not later ones). That is the date that the IANA will tell Regional Internet Registries like ARIN, RIPE and APNIC, that there are no more to replenish their supplies. The RIRs will likely have enough on hand to last another six months at most. I have personally joined APNIC as a member and reserved a "/22" block of IPv4 addresses (a little over 1000 of the precious, and increasingly scarce addresses for the First Internet). These will cost me about 1000 USD per year, but I will be able to continue running my companies and other activities for many years to come. You can think of this as staking out some of the last remaining lots in a virtual Oklahoma Land Rush. I am also doing this in order to obtain my very own "/32" block of the shiny new IPv6 addresses. You can think of this as getting an enormous spread of prime real estate in the virtual New World of the Second Internet. Anyone that wants to today can do the same thing (at least for a little while longer). I understand what's coming, and I know what I'll be able to build on that prime real estate. I think it's a hell of a bargain.

1.3 – So Just What Is It That We Are Running Out Of?

There is a great deal of confusion and misunderstanding about this, as important as it is. Many people think that "internet addresses" are things like *www.ipv6.org*. That is not an Internet Address, it is a *symbolic nodename*. That is an important part of a URI (Uniform Resource Identifier), which adds things such as a protocol designator (e.g. http:, mailto: or sip:), possibly a non-standard port number (e.g. ":8080") and often a file path (e.g. "/files/index.html"). If you allowed up to 30 characters for a nodename (the preceding example being 14 characters long) and allowed any alpha or numeric character and the hyphen (a-z and 0-9 and "-"), which are all legal in Internet nodenames, this would give a total of 37 possible characters in each position. That means there are 37^{30} (1.11×10^{47}) possible nodenames, although most of them would be *really* obscure and hard to remember, like

poas5jdpof343jijio.iuhiu3hu4ifer.com. That's a *lot* of names. There is still a staggering number of names that are easy to remember. More than could ever be used in the next hundred years. So just what is it that we are running out of?

The nodenames that you (and most humans) use to specify a particular node on the Internet, like *www.ipv6.org*, are made possible by something called the Domain Name System (DNS). Those nodenames are not used in the actual packets as source and destination addresses (see section on IPv4 addressing model for the gory details). The addresses used in the packets on the wire in the First Internet are 32 bit binary numbers. These are usually represented for us slow and stupid humans in *dotted decimal notation* like 123.45.67.89. With a 32 bit address, there are 2^{32} (about 4.3 billion) distinct values. When you use a *symbolic nodename* (known technically as a *Fully Qualified Domain Name*, or FQDN) in an application, that application sends it to a DNS server, which returns the numeric IP address associated with it. *That's* the address that is used in packets on the wire, for routing the packet to its destination. The DNS nodenames are like the names of people you call, the IP addresses are like their phone numbers. DNS is like an online telephone book that looks up the "phone number" (IP address) for "people" (nodes) you want to "call" (connect to). Did you know that you can surf to IP addresses? Try entering the URL *http://15.200.2.21*. That's a whole lot harder to remember than *www.hp.com*, which is why DNS was invented. It's these 32 bit numeric addresses (that most people never see) that we are running out of. The good news is that you can keep typing *www.hp.com*, and DNS will soon return both the old style 32 bit IPv4 address and a new style 128 bit IPv6 address, which will be put into IPv6 packets. Given the choice, your applications will prefer to use the new IPv6 address. You will hardly notice the difference, unless you are a network engineer or a network software developer. Except there's going to be an awfully lot of cool new stuff to do, and new ways of doing old things, plus the Internet is going to work better than it ever has.

Can you imagine trying to manage today with 5 digit telephone numbers? In a few years, that's what IPv4 is going to feel like.

1.4 – But You Said There Were 4.3 Billion IPv4 Addresses?

But, I hear you protest, there are only 1.3 billion nodes currently connected to the Legacy Internet, and there are 4.3 billion possible IPv4 addresses. Aren't there still some 3 billion addresses left? Well, no, sad to say, there aren't.

On February 15, 2010 (when I started writing this book), there were only 364 million addresses left to assign (again, from the NetCore countdown clock). On May 12, 2010 (3 months later), there were only 298 million addresses left. *What the heck happened to the rest?* Well, when the First Internet was being rolled out, there were about 600 nodes in the world, and 4.3 billion looked a lot like "infinity" to the people involved. So, giant chunks of addresses were generously given out to early adopter organizations. For example, M.I.T. and HP were given "class A" blocks of addresses (about 16.7 million addresses, or 1/256 of the total address space, each). Smaller organizations were given "class B" blocks of addresses (each having about 65,535 addresses). Most of these organizations are not using anywhere *near* all of those addresses, but they have never been willing to turn them back in. As detailed in the Organization for Economic Co-operation and Development (OECD) study on IPv4 address space depletion and migration to IPv6, it is *very* difficult and time consuming to recover these "lost" addresses. Also some blocks of addresses were used for things like multicast, experimental use and other purposes.

We are getting more efficient in our allocation of IPv4 addresses, but even with every trick we know, they will likely all be gone by March 2012, *or before*. It is easy to measure how quickly IP addresses are being allocated, and how many are left, so it's not exactly rocket science to predict when they will run out. *These projections assume there will be no "bank run" or panic allocations as we get near the bottom of the barrel, or increases in the rate that addresses are allocated.* Both of these assumptions are really optimistic. That's why I keep saying "or sooner". The people of Taiwan have announced their intention to connect some 3 *billion* devices to the Internet in the next few years. Even if we gave them all 298 million of the remaining addresses, they still could not connect that many devices. They can only do this by going to longer IP addresses (hence a larger address space). This is one of the main things that IPv6 is about.

1.5 – Is IPv6 just an Asian Thing?

I have heard many comments from U.S. networking professionals and Venture Capitalists that IPv6 is an "Asian thing", something that is of little interest or concern to Americans. This shows an unusually provincial view of an extremely serious situation. This attitude is only partly due to the inequitable distribution of addresses for the First Internet (there are over 6 IPv4 addresses per American citizen, compared to only about 0.28 per person for the rest of the world). It has a lot more to do with a lack of knowledge of how certain parts of the First Internet really work, compounded by a limited time horizon compared to Asian businessmen, who routinely plan 5 to 10 years ahead. American business schools teach that nothing is important beyond the next quarter's numbers. The depletion of IPv4 addresses is beyond the end of next quarter, *but not by very much*. Expect a major panic when the IPv4 depletion date comes within the time horizon of American businessmen ("why didn't you *warn* us about this?").

Any country or organization that (for whatever reason) doesn't migrate to IPv6 is going to still be "riding horses" while the rest of us are zipping around in these newfangled "cars". I have nightmares about the U.S. being just as reluctant to go to IPv6 as they were to adopt the metric system (the U.S. is the only industrialized country *not* to have adopted the metric system, and I doubt they ever will). They *could* decide to stay with IPv4. If so, it will become increasingly difficult for them to connect to non-U.S. websites, or for people in other countries to connect to U.S. websites. It will impact all telephone calls between the U.S. and anywhere else in the world. It will make IT products designed for the U.S. market of little interest outside of the U.S. (kind of like automobiles that can't be maintained with metric tools). This will isolate the U.S. even further, and essentially leave leadership in Information Technology up for grabs. Japan, China and South Korea are quite serious about grabbing that leadership, and they are well along their way to accomplishing this, by investing heavily in IPv6 for several years already.

Being good engineers, while the IETF has the "streets dug up" increasing the size of IP addresses, they are fixing and enhancing many of the aspects of IPv4 (QoS, multicast, routing, etc.) that weren't done quite as well as they might have been (who could have envisioned streaming video 27 years ago?). IPv6 is not just bigger addresses. It's a whole new and remarkably robust platform on which to build the Second Internet.

1.6 – So What is This "Second Internet"?

Most things in computer technology evolve through various releases or generations, with significant new features and capabilities in the newer generations. For example, 2G, 2.5G and 3G cell phones. The Internet is no exception. The remarkable thing though, is that the first generation of the Internet has lasted for 27 years already, and we are only now coming to the second generation of it. There are a number of technology trends going on right now, and some of them have been hyped heavily in the press. Some of them sound a lot like they might be the next generation of the Internet. Let's see if we can narrow down what I mean by "the Second Internet" by discussing some the things that it is *not*.

1.6.1 – Is the *Next Generation Network* (NGN) that Telcos Talk About, the Second Internet?

Telcos around the world have been moving towards something they call NGN for some time. Is that the same thing as the Second Internet? Well, there is certainly a lot of overlap, but no, NGN is something quite different.

Historically, telephone networks have been based on a variety of technologies, mostly *circuit switched*, with call setup handled by SS7 (Signaling System 7). The core of the networks might be digital, but almost the entire *last mile* (the part of the telco system reaching from the local telco office into your homes and businesses) is *still* analog today. There was some effort at upgrading this last mile to digital with ISDN (Integrated Services Digital Networks), but some terrible decisions regarding tariffs (the cost of services) pretty much killed ISDN in many countries, including the U.S.

The ITU (International Telecommunication Union), an agency of the United Nations that has historically overseen telephone systems worldwide, defines NGN as packet-switched networks able to provide services, including telecommunications, over broadband, with Quality of Service enabled transport technologies, and in which service-related functions are independent from underlying transport-related technologies. It offers unrestricted access by users to different telecommunication service providers. It supports generalized mobility which will allow consistent and ubiquitous service to users.

In practice, telco NGN has three main aspects:

- In telco core networks, there is a consolidation (or *convergence*) of legacy transport networks based on X.25 and Frame Relay into the data networks based on TCP/IP (still, alas, mostly TCP/IPv4 so far). It also involves moving from circuit switched (mostly analog) voice technology (the Public Switched Telephone Network, or PSTN) to Voice over Internet Protocol (VoIP). So far, the move to VoIP is mostly internal to the telcos. What is in your house and company is good old POTS (Plain Old Telephone Service).
- In the "last mile", NGN involves migration from legacy split voice and data networks to Digital Subscriber Line (DSL), making it possible to finally remove the legacy voice switching infrastructure.
- In cable access networks, NGN involves migration of constant bit rate voice to Packet Cable standards that provide VoIP and Session Initiation Protocol (SIP) services. These are provided over DOCSIS (Data Over Cable Service Interface Specification) as the cable data layer standard. DOCSIS 3.0 does include good support for IPv6, though it requires major upgrades to existing infrastructure. There is also a "DOCSIS 2.0 + IPv6" standard which supports IPv6 even over the older DOCSIS 2.0 framework, typically requiring only a firmware upgrade in equipment. That will likely get rolled out before DOCSIS 3.0 can be (DOCSIS 3.0 requires hardware upgrades).

A major part of NGN is **IMS** (the *IP Multimedia Subsystem*). To understand IMS, I highly recommend the book "The 3G IP Multimedia Subsystem (IMS) – Merging the Internet and the Cellular Worlds", by Gonzalo Camarillo and Miguel A. Gaccia-Martin. This was published by John Wiley & Sons, in 2004. This book says that IMS (which is the future of all telephony) was designed to work *only* over IPv6, using DHCPv6, DNS over IPv6, E.164 Number Mapping (ENUM), and Session Initiation Protocol/Real-time Transport Protocol (SIP/RTP) over IPv6. IMS is *so* IPv6 specific, that some of the primary concerns are how legacy IPv4-only SIP based user agents (hardphones and softphones) will communicate with the IPv6 core. One approach is to use dual-stack SIP proxies that can in effect translate between SIP over IPv4 and SIP over IPv6. Translation of the media component (RTP) is a bit trickier, and will be handled by Network Address Translation between IPv4 and IPv6. Newer IPv6 compliant user agents will be able to interoperate directly with the IMS core, without any gateways, and solve many problems. They are beginning to appear. In my home today, I am using some dual stack IP phones from a great Korean company named Moimstone.

The first "Internet over telco wireless service" in early 2G networks was **WAP** (Wireless Application Protocol). WAP 1.0 was released in April 1998. WAP 1.1 followed in 1999, followed by WAP 1.2 in June 2000. The Short Messaging System (SMS) was introduced but only IPv4 was supported. Speed and capabilities were somewhat underwhelming.

2.5G systems improved on WAP with **GPRS** (General Packet Radio Service), with theoretical data rates of 56 to 114 Kbits/sec. GPRS included "always on" Internet access, Multimedia Messaging Service (MMS), and Point-to-point service. It increased the speed of SMS to about 30 messages/sec. Even Filipinos can't text *that* fast. As with WAP, only IPv4 was supported.

2.75G systems introduced **EDGE** (Enhanced Data Rates for GSM Evolution), also known as EGPRS (Enhanced GPRS). EDGE service provided up to 2 Mbit/sec to a stationary or walking user, and 348 Kbit/sec in a moving vehicle. IPv6 service has been demonstrated over EDGE, but is not widely deployed.

3G systems introduced **HSPA** (High Speed Packet Access), which consisted of two protocols, **HSDPA** (High Speed Downlink Packet Access) with theoretical speeds of up to 14 Mbit/sec service, and **HSUPA** (High Speed Uplink Packet Access) with up to 5.8 Mbit/sec service. Real performance was again somewhat lower, but better than with EDGE. HSPA had good support for IPv6.

The last gasp for 3G (sometimes called "3.9G") is **LTE** (Long Term Evolution). LTE is completely based on IP, and primarily (but as of recent versions of the 3GPP specification, no longer exclusively) based on IPv6. Earlier versions of the specification clearly described it with IPv6 mandatory and IPv4 support optional. It has now been reworded to make most aspects "IPv4v6" (dual-stack). 3G was still based on two parallel infrastructures (circuit switched and packet switched). LTE is packet switched *only* ("All IP"). There are a few deployments of LTE (some of which are described incorrectly as "4G") around the world.

4G systems (now being designed) complete the transition to all IP and even higher speed wireless transports. They will use an all-IP infrastructure for both wired and wireless. The specification for 4G claims peak downlink rates of at least 100 MBit/sec, and uplink of at least 50 Mbit/sec. 4G requires a "flat" IP infrastructure (no NAT), which can only be accomplished with IPv6. IPv4 address space depletion will happen before 4G is rolled out, so IPv4 is not even an option. IPTV is a key part of 4G, which requires fully functional multicast, scalable to very large customer bases. That also requires IPv6.

So, clearly the Telco's NGN is moving more and more *towards* IPv6 in the near future, but current deployments are still mostly IPv4. However, NGN is just as clearly *not* the Second Internet described in this book. You might say that NGN (once it reaches 4G) will be just another one of the major applications hosted on the Second Internet, peer to E-mail, the web, IPTV, etc.

There will be *much more* to the Second Internet than just telephony, including most broadcast entertainment, exciting new possibilities for non-telephonic communication paradigms (fully decentralized instant messaging and peer-to-peer collaboration), smart building sensor and control systems, and ubiquitous connectivity in essentially all consumer electronics, including MP3 players, electronic book readers, cameras and personal health monitoring. It will also impact automotive design. See www.car-to-car.org for some exciting new concepts in "cooperative Intelligent Transport Systems" that depend heavily on IPv6 concepts such as Networks in Motion (NEMO) defined in Request for Comments (RFC) 3963, and ad-hoc networks. In fact, *only* IPv6 is being used in their designs, although it is a slightly modified version of IPv6 that is missing some common functionality such as Duplicate Address Detection. Their modified IPv6 runs on top of a new, somewhat unusual Link Layer called the C2C Communication Network, which itself is built on top of IEEE 802.11p, also known asWireless Access in Vehicular Environments (WAVE).

1.6.2 – Is *Internet2* the Second Internet?

Internet2 is an advanced academic and industrial *consortium* led by the research and education community, including over 200 higher education institutions and the research departments of a number of large corporations. They have deployed a world-wide research network called *the Internet2 Network*. While IPv6 is definitely being used on the Internet2 network, their scope goes well beyond IPv6, in such areas as network performance. The first part of the Internet2 network (called *Abilene*) was built in 1998, running at 10Gbit/sec, even over Wide Area Network (WAN) links. It was associated with the National Lambda Rail (NLR) project for some time. Internet2 and NLR have since split and moved forward along two different paths. Today, most links in the global Internet2 network are running at 100Gbit/sec. This is over 1000 times faster than typical WAN links used by major corporations today. It is even 10 to 100 times faster than state of the art *LANs* (Local Area Networks).

Internet2 also features advanced research into secure identity and access management tools, on-demand creation and scheduling of high-bandwidth, high-performance circuits, layer 2 Virtual Private Networks (VPNs) and dynamic circuit networks (DCNs).

A recent survey of Internet2 sites showed that only a small percentage of them have even basic IPv6 functionality deployed, such as IPv6 DNS, E-mail or VoIP over IPv6.

Essentially Internet2 is primarily concerned more with extreme high-end performance (100Gbit/sec and up), and very advanced networking concepts not likely to be used in real-world systems for decades. Although they do profess support for IPv6, they have not aggressively deployed it, and it is definitely not central to their efforts. They are doing little or no work on IPv6 itself, or in new commercial applications based on IPv6. I guess those areas are not very exciting to academicians.

The real world *Second Internet* I am writing about in this book will be built primarily with equipment that mostly has the same performance as current First Internet sites (no more than 100Mbit/sec on WAN links for some time to come and only that high in advanced countries). In much of the world today 1.5Mbit/sec to 10Mbit/sec is considered good. Maybe 100Gbit/sec will be widely deployed by 2030 to 2040, but ultra high performance is not necessary to provide the revolutionary benefits described in this book. To give you an idea, Standard Definition (SD) TV requires about 2Mbit/sec bandwidth per channel, and High Definition (HD) TV requires about 7 to 10 MBit/sec bandwidth per simultaneously viewed channel. That is about the most bandwidth intensive application you will likely see for most users for some time to come. Voice only requires about 8 to 64 Kbit/sec for good quality. In Japan and Korea today, home Internet accounts typically have about 50 to 100 Mbit/sec performance. In my hotel room in Tokyo recently, I measured 42 Mbit/sec throughput. That is enough for almost any use today. Most users, even in companies, would be really challenged to make effective use of 100 Gbit/sec bandwidth. With that bandwidth you could download the entire Encyclopedia Britannica in just a few seconds (including images) or a typical Blu-Ray movie (about 25 GBytes) in about 2 seconds.

The necessary equipment and applications for the Second Internet can in many cases be created with software or firmware upgrades (except for older and low-end devices that don't have enough RAM or ROM to handle the more complex software, and in high end telco and Internet Service Provider (ISP) level products that include hardware acceleration). If you look at the products created by my company (*www.infoweapons.com*), you will see what I think are some of the most important components that are needed to build the Second Internet: a dual stack DNS/DHCP appliance, a dual stack firewall with 6in4 tunneling, and a dual stack VoIP server (IP PBX). We will soon be releasing a dual stack network monitoring appliance as well.

The main technical advantages of the Second Internet will not be higher bandwidth, but the vastly larger address space, the restoration of the flat address space (elimination of NAT), and the general availability of working multicast. All of these are made possible by migration to IPv6, which involves insignificant costs compared to supporting 100Gbit/sec WAN links. Perhaps generally available WAN bandwidth in that range will be what characterizes the *Third Internet*.

You can find out more about Internet2 on their homepage, *http://www.internet2.edu*.

So, Internet2 (despite the name) is *not* the Second Internet I am writing about. Internet2 is primarily an academic exercise that will not bear fruit for many decades. What they are doing is very important in the long run, but it does not address, and will not solve, the really major problems facing the First Internet today. The Second Internet is being rolled out today, and will be largely functional before the last IPv4 address is given out by the RIRs, probably sometime in 2011. That event will mark the end of the First (IP4-only) Internet.

1.6.3 – Is *Web 2.0* the Second Internet?

First, if you think that the terms "World Wide Web" and "Internet" are synonymous, let me expand your worldview a bit, in the same way that Copernicus did for people's view of our Solar System back in the mid 1500s. The "World Wide Web" is basically *one* service that runs on a much larger, more complex thing which is called the *Internet*. The web is a simple *client-server* system based on HTTP (HyperText Transfer Protocol) and HTML (HyperText Markup Language). Due to extremely serious limitations and

inefficiencies of these standards, both have been enhanced and extended numerous times. The result is still not particularly elegant to real network software designers or engineers, but it has *clearly* had a major impact on the world. The technology of the web was a refinement and convergence of several ideas and technologies that were in use before HTML and HTTP were created by Tim Berners-Lee in the late 1980s, at CERN. But there is lot to the Internet beyond the web (E-mail, instant messaging, video conferencing, VoIP, file transfer, peer-to-peer (P2P), VPNs, IPTV, etc.). There are thousands of Internet protocols, of which the web uses *two* (HTTP and HTTPS).

HyperText, WAIS/SGML and Gopher

The terms HyperText and HyperMedia were coined by Ted Nelson in 1965, at Brown University. These terms referred to online text documents (or rich media, including pictures, sound, and other media content) that contained *links* that allowed building paths from any word or phrase in the document to other parts of the same document, or parts of other documents that were also online. In August 1987, Apple Computer released the first commercial HyperText based application, called HyperCard, for the Macintosh. There were already document storage and retrieval systems on the early Internet, such as WAIS (Wide Area Information Server). WAIS was based on the ANSI Z39.50:1988 standard, and was developed in the late 1980s by a group of companies including Thinking Machines, Apple Computer, Dow Jones and KPMG Peat Marwick. As with the web, there were both WAIS servers and clients. A later version of WAIS was based on ANSI Z39.50:1992, which included SGML (Standard Generalized Markup Language, ISO 8879:1986) for more professional looking documents. There was another Internet application called Gopher (University of Minnesota, circa 1991) that could distribute, search for, and retrieve documents. Gopher was also primarily text based, and imposed a very strict hierarchical structure on information.

HTML and HTTP

Tim Berners-Lee combined these three concepts (HyperText, WAIS/SGML and Gopher document retrieval) to create HTTP and HTML. HTML was a very watered down and limited markup language compared to SGML. SGML is capable of created highly sophisticated, professional looking books. In comparison, HTML allows very limited control over the final appearance of the document on the client's screen. HTTP was a very simple protocol designed to serve HTML documents to HTTP client programs called *web browsers*. A basic HTTP server can be written in one afternoon, and consists of about half a page of the C programming language (I've done it, and retrieved documents from it with a standard browser). The first browser (Lynx, 1992) was very limited (text only, but including hypertext links). In 1993, at the National Center for Supercomputing Applications (NCSA) at the University of Illinois, the first Mosaic web browser was created (running on X Windows in UNIX). Because it was created for use on X Windows (a platform with good support for computer graphics), many graphics capabilities were added. With the release of web browsers for PC and Macintosh, the number of servers went from 500 in 1993 to 10,000 in 1994. The *World Wide Web* has since grown to millions of servers and many versions of the web client (Internet Explorer, Mozilla Firefox, Safari, Opera, Chrome, etc.). It's been so successful that a lot of people today think that the World Wide Web *is* the Internet.

Web 2.0

The term *Web 2.0* was first coined by Darcy DiNucci in 1999, in a magazine article. The current usage dates from an annual conference that began in 2004, called "Web 2.0", organized and run by Tim O'Reilly (owner of O'Reilly Media, publisher of many excellent computer books).

Many of the promoters of the term Web 2.0 characterize what came before (which they call *Web 1.0*) as being "Web as Information Source". Web 1.0 is based on technologies such as PHP, Ruby, ColdFusion, Perl, Python and ASP (Active Server Pages). In comparison, Web 2.0 is "Network as Platform", or the "participatory web". It uses some or all of the technologies of Web 1.0, plus new things such as Asynchronous JavaScript, XML, Ajax, Adobe Flash and Adobe Flex. Typical Web 2.0 applications are the *Wiki* (and the world's biggest Wiki, the *Wikipedia*), blogging sites, social networking sites like *FaceBook*, video publishing sites like *YouTube*, photographic snapshot publishing sites like *Flickr*, Google Maps, etc.

Andrew Keen (British-American entrepreneur and author) claims that Web 2.0 has created a cult of digital narcissism and amateurism, which undermines the very notion of expertise. It allows *anyone anywhere* to share their own opinions and content, regardless of their talent, knowledge, credentials, or bias. It is "creating an endless digital forest of mediocrity: uninformed political commentary, unseemly home videos, embarrassingly amateurish music, unreadable poems, essays and novels." He also says that Wikipedia is full of "mistakes, half truths and misunderstandings". Perhaps Web 2.0 has made it *too* easy for the mass public to participate. Tim Berners-Lee's take on Web 2.0, is that it is just a "piece of jargon". In the finest tradition of Web 2.0 these comments, which were found in the Wikipedia article on Web 2.0, probably include some mistakes, half-truths and misunderstandings.

Basically, Web 2.0 does not introduce any revolutionary new technology or protocols; it is more a refinement of what was already being done on the web, in combination with a new emphasis on end-users becoming not just passive consumers, but also producers of web content. The Second Internet will actually help make Web 2.0 work better, as it removes the barriers that have existed in the First Internet since the introduction of NAT to anyone becoming a producer of content. If anything, on the Second Internet, these trends will be taken even further by decentralizing things. There will be no need for centralized sites like YouTube or Flickr to publish your content, just more sophisticated search engines or directories that will allow people to locate content that will be scattered all over the world. Perhaps that will be the characterizing feature of *Web 3.0*?

Web 2.0 is a really minor thing compared to the Second Internet. What isn't pure marketing hype is an evolutionary development of *one* of the major services (the *World Wide Web*) out of perhaps a dozen that the Second Internet will be capable of hosting. These include global telephony, newer forms of communication like decentralized instant messaging, major new Peer to Peer applications (not just file sharing), global broadcast entertainment via multicast IPTV, connectivity between essentially all consumer electronic products, personal healthcare sensor nets, smart building sensor nets, etc.

1.7 – Whatever Happened to IPv5?

Two of the common questions people ask when they start reading about IPv6 is "If it's the next version after *IPv4*, why isn't it called *IPv5*?" and "What happened to the *first three* versions of IP?"

There is a four bit field in every IP packet header that contains the IP version number in binary. In IPv4, that field contains the binary value 0100 (4 in decimal) in every packet. An earlier protocol (defined in

RFC 1190, “Experimental Internet Stream Protocol, Version 2 (ST-II)”, October 1990) used the binary pattern 0101 (5 in decimal) in the IP version field of the packet header. The Internet Stream Protocol was not really a replacement for IPv4, and isn’t even used today, but unfortunately the binary pattern 0101 was allocated to it. The next available bit pattern was 0110 binary (6 in decimal). It would be even more embarrassing than explaining that there was no IPv5, to explain why the IP Version Number field for IPv5 contained the value 6. Now you know.

So what did happen to IPv1, IPv2 and IPv3? Well, TCP went through three versions (including all the functionality of IP) before IP was split out into a separate protocol in RFC 791. So, IP began its independent existence at *version 4* (kind of like Windows NT starting life at version 3.1).

No protocols called IPv1, IPv2, IPv3 or IPv5 ever existed. *IPv4* was the first release of the Internet Protocol (1G Internet), and *IPv6* is the second release (2G Internet). Hence my name for the Internet based on it: the *Second Internet*.

There have been rumors about an *IPv9* protocol in China. A Venture Capital firm in Hong Kong actually asked me if China was already that far ahead of the rest of the world, and shouldn’t we be supporting their version? It seems some researcher in a university there published a paper on an “IPv9”, but it was never implemented, and wasn’t a replacement for IPv4 (let alone IPv6) anyway. It was a way to use 10 digit decimal phone numbers in a modified DNS implementation instead of alphanumeric domain names, for all nodes on the Internet. I guess if you speak only Chinese, a 10 digit numeric string may seem easier to use than an English domain name using Latin characters. Fortunately for Chinese speakers, we will soon have *Internationalized Domain Names* in Chinese and other languages.

Actually, there is a real RFC about IPv9, which you might enjoy reading. See RFC 1606, “A Historical Perspective On The Usage of IP Version 9”, *April 1*, 1994. This has nothing to do with the Chinese IPv9, and is much funnier. This RFC is a network engineer’s equivalent to an integrated circuit data sheet I once saw, concerning a logic gate (circuit) called a “maybe gate”. A “maybe gate” is similar to an “or gate”, “and gate” and “nand gate”. There were two inputs to the maybe gate, each of which could be logic 0 or logic 1. The output was “maybe 1, maybe 0”, it all depended on how the gate *felt* right then. Please notice the release date of RFC 1606.

1.8 – Let’s Eliminate the Middle Man

One of the things that the Second Internet does better than anything is *disintermediation*. Just as E-mail eliminated the need for a central Post Office, the features of the Second Internet will eliminate the need for many existing centralized organizations and services. With a real decentralized end-to-end connectivity model, there is no need for two users to connect to a central server (such as AOL’s Instant Messenger Service) in order to chat with each other. They will simply connect directly to each other. That’s hard to do today, because of NAT.

The restoration of the original (pre-NAT) flat address space, and the plethora of addresses will allow anyone or anything to connect directly to anyone or anything on the Second Internet. It’s going to be a *very* different online world. Many business models will go by the way, and many new ones will explode on the scene and make some new entrepreneurs very wealthy. Someone will need to provide

centralized directory and presence servers that will let people locate each other, so that they *can* connect directly to each other.

A number of years ago, a gentleman in my previous home town of Atlanta, Georgia (home to Coca-Cola, and UPS) had a small UHF TV station (WTBS, Channel 17) that mostly broadcast old movies and Atlanta Braves baseball, both of which he loved. He was one of the first people to realize that he could relay his TV station's signal through a transponder on a geo-stationary satellite ("that's a really tall broadcast antenna"), and the rest is history. The man was Ted Turner, and his bright idea created the Turner Broadcast System (TBS), which along the way produced CNN, CNN Headline News, Cartoon Network, Turner Network Television (TNT), and many other things. His success allowed him to *buy* the Braves baseball team, and the entire film library of MGM (not to mention a famous starlet wife, sometimes also called "Hanoi Jane"). When he began relaying his Channel 17 signal, his viewership went from maybe 10,000 to 10,000,000 virtually overnight. That was a world changing insight.

Some bright entrepreneur is going to realize that global multicast IPTV is the same kind of opportunity. Wonder what he (or she) will create with the wealth thereby generated? What country will they be from? I'm betting India.

I did warn you that this is revolutionary, highly disruptive technology. However, with great disruption comes great opportunity.

1.9 – Why Am I the One Writing This Book? Just Who Do I Think I Am, Anyway?

I have been personally involved in helping create and deploy the Second Internet for many years. I've spoken at IPv6 summits around the world, including Beijing, Seoul, Kuala Lumpur, Manila, Taipei, Potsdam and Washington D.C. I have so far invested 8 years of my life and about $9M of my own personal funds (which came from selling a previous Internet based venture called CipherTrust where I was cofounder). I have relocated myself and my family from the United States to Asia to be where the IPv6 action is (in these early days, the Second Internet really *is* somewhat of an "Asian thing", but soon enough it will be worldwide). I've built a company that now has six years of experience, deep pools of expertise, and already three critical products needed to build networks for the Second Internet (with two more products underway). We have recently been certified as one of the six official IPv6 Ready testing centers in the world. The transition to the Second Internet is by far the biggest business and technology opportunity I've seen in my 37 years in IT.

Now, *have I gotten your attention?* Great – now let's explore just what the Second Internet is all about.

Chapter 2 – History of Computer Networks Up to TCP/IPv4

A long time ago (in a Galaxy not too far away), people started connecting computers together. A few brave souls tried to do this with dial-up 1200 baud modems over phone lines. Pioneers brought up Bulletin Board Systems (message boards that one person at a time could dial into and exchange short messages, and later small files, with each other). I brought up the 8th BBS in the world, in Atlanta, about 1977, using code from the original CBBS in Chicago (created by Ward Christensen and Randy Suess) and a modem donated by my friend Dennis Hayes (of Hayes Microcomputer Products). Later there were thousands of online Bulletin Board Systems, all over the world. Soon there followed commercial "information utilities" like CompuServe and The Source, which were like giant BBS'es with many more features. Tens of thousands of users could connect to these simultaneously. It was like the first crude approximation to the Internet of today, based on circuit switched connections over telephone lines. Everything was text oriented (non-graphical) and very slow. 1200 bits/sec was typical at first, although later modems with speeds of 2400 bits/sec, 9600 bits/sec, 14.4KB/sec, 28.8KB/sec and finally 56KB/sec were developed and came into widespread use. Later these modems were primarily used to dial into an ISP to connect to the Internet, and some people are still using them this way.

2.1 – Real Computer Networking

While home computer users were playing around with modems and bulletin board systems, the big computer companies were working on ways to connect "real" computers at higher speeds, and with much more complex software.

2.1.1 – Ethernet and Token Ring

Much of this was based on Ethernet, which was created by a team at Xerox Palo Alto Research Center (PARC) led by Robert Metcalf between 1973 and 1975. The first specification (1976) ran at 3 Mbit/sec. Metcalf left PARC in 1979 to create 3com and create commercial products based on Ethernet. Working together with Digital Equipment Corporation (DEC), Intel and Xerox (hence the "DIX" standard), 3com released the first commercial products running at 10 MBit/sec. Ethernet was standardized in 1980 by the IEEE (Institute for Electrical and Electronic Engineers) as 802.3. Early versions ran on 10base2 (a small diameter coax cable) or 10base5 (a larger diameter coax cable). These used a "multidrop" architecture, which was subject to many reliability problems. With the introduction of the simpler to deploy and manage "unshielded twisted pair" (actually 4 pair, or 8 wire) cables (known as 10baseT, mid 1980s), and star architectures using "hubs" and later "switches", local area networks really took off. Today, virtually all Ethernet networks use twisted pair copper wire (up to gigabit speed) or fiber optic cable (for higher speed and longer runs). I helped deploy a 10base2 coax Ethernet network in Hong Kong in 1993. Trust me, twisted pair cabling is a *lot* easier.

IBM for many years pushed a competing physical layer network standard called "Token Ring" (later standardized as IEEE 802.5). Token Ring was available in 4 Mbit/sec and 16 Mbit/sec versions. Later, a

100 Mbit/sec version was created, but by then Ethernet dominated the market and Token Ring quietly disappeared. FDDI (Fiber Distributed Data Interface) still in use today is based on the token ring concept.

2.1.2 – Network Software

Network software quickly evolved once Ethernet and Token Ring hardware became available. One of the main goals was to "hide" the differences between various hardware level technologies (Ethernet, Token Ring, Wireless, etc.) from the higher level software. This led to the multiple layers of the network stack. The bottom layer is very hardware specific, and the upper layers introduce more and more hardware independence, so that applications can be written once, and run over any hardware transport.

DEC was one of the first companies to create networking software with DECNET (1975). IBM had System Network Architecture (SNA, 1974). Xerox created the PARC Universal Packet protocol (PUP, late 70's) which eventually evolved into Xerox Network Services (XNS, early 80's) at PARC. XNS was the basis for the later Banyan Vines Network OS, based on "Vines IP" (similar to but incompatible with IPv4 from TCP/IP). Banyan Vines included the first network directory service, called "StreetTalk". XNS also was the basis for Novell Netware (IPX/SPX, 1983), which eventually added its own Netware Directory Services (NDS, 1993). NetWare did not fully support TCP/IP until 1998, which allowed Microsoft (who supported TCP/IP first) to take over leadership in personal computing networks.

Microsoft worked with 3com to create their own network OS, called LAN Manager. It used SMB (Server Message Block) protocol on top of either NBF (NetBIOS Frames Protocol) or modified XNS. In 1990, Microsoft added support for TCP/IP as an alternate protocol (LAN Manager 2.0). With the release of Windows NT Advanced Server in 1993, Microsoft finally phased out LAN Manager. By Windows NT v3.51 (May 1995) Microsoft encouraged users to deploy *only* TCP/IP (four years ahead of Novell's support for TCP/IP). This lead time allowed Microsoft to take over leadership in personal computer networks from Novell. Microsoft introduced their version of network directory services in Windows Server 2000, now known as Active Directory. The SMB protocol still survives as Microsoft's "File and Printer Sharing" protocol (now layered on TCP/IP, instead of NetBIOS or XNS). An Open Source implementation of this is available as SAMBA.

2.2 – The Beginnings of the Internet (ARPANET)

While all this commercial activity was going on, the U.S. Military (at their Advanced Research Projects Agency, or ARPA), with the help of Bolt, Beranek and Newman (BBN) and Mitre, were designing a new, decentralized communication system based on packet switching. Existing communication systems (telephone, radio, etc.) were *centralized*, and hence subject to being completely disabled due to the failure or loss of a few central nodes. Packet switched networks were highly *decentralized*, and had a fascinating new property, which is that you could lose large parts of a network, and the remaining parts would still work (assuming at least some links connected the working parts).

The first network protocol developed as part of ARPANET was called the 1822 protocol (named after BBN report 1822), and was implemented by a Network Control Program, so the protocol was often referred to as NCP. The first E-mail was sent over NCP in 1971, and the File Transfer Protocol followed in 1973. On Jan 1, 1983 ("flag day"), NCP was turned off officially, leaving only TCP/IPv4 on the Internet.

You might think of the NCP era as phase 1 of the First Internet, with the IPv4 era being phase 2 of the First Internet. Otherwise the new Internet based on TCP/IPv6 will be THIRD Internet. Fortunately, there is no need for a flag day to go from TCP/IPv4 to TCP/IPv6, as they can co-exist (and probably will for perhaps 5 to 10 years).

In May 1974, Vint Cerf and Bob Kahn released the paper "A Protocol for Packet Network Interconnection". This described a monolithic protocol called TCP that combined the features of both modern TCP and IPv4. Later John Postel was instrumental in splitting apart TCP and IP as we know them today. Vint Cerf is today considered the "father of TCP/IP", and is now the IPv6 Evangelist at Google. He understands very well the problems with the current implementation of TCP/IPv4 (and why these things were done). He advocates for users to migrate to TCP/IPv6, which restores his original concept of a flat address space (no NAT), where any node can connect directly to any other node.

If you'd like to read about the creation of the First Internet, I recommend the book "Where Wizards Stay Up Late: The Origins of the Internet", by Katie Hafner and Matthew Lyon. It is of considerable interest to those of us creating the Second Internet, as we facing some of the same problems they did. Only this time around, we've got over a billion legacy users (and staggering investments in hardware and software) to worry about. On the other hand, we've got three decades of operational experience with TCP/IPv4 to draw upon.

Higher level software protocols were built on top of the TCP and IP layers, called "application protocols", such as SMTP (for E-mail), FTP (for file transfer) and Telnet (for terminal emulation) and more recently HTTP (used in the Web), and SIP & RTP (used in VoIP). The resulting suite of protocols became known for its two most important protocols, TCP and IP, or TCP/IP.

2.2.1 – UNIX

About this time (1973), Bell Labs (a research group within AT&T) created an interesting new operating system (called PWB-UNIX) and a new language (in which UNIX was written) called "C". Because of a 1958 consent decree, AT&T as a regulated monopoly was not allowed to market or sell UNIX commercially. They licensed it (complete with source code) to a number of universities. One of these was the University of California at Berkeley (also famous for being the center of communist student activities at the time). The team at UCB extended UNIX in several very important ways such as adding Virtual Memory. They also integrated the new network protocol from ARPA as the first commercial implementation of TCP/IP. The "Berkeley System Distribution" of UNIX became a main branch. Over time, they rewrote most of it and wanted to release it for free. AT&T sued them in court, and it seems most of the examples of "stolen code" AT&T cited had actually been written at UCB. The judge ruled that if UCB rewrote the remaining 10% or so (so there was zero original AT&T code), they could release that. That rewrite became 386BSD, the starting point for FreeBSD (the first Open Source operating system). Interestingly enough, FreeBSD was chosen by Japan's KAME project to deploy the first version (the "reference" implementation) of an IPv6 stack, in an eerie echo of BSD UNIX's choice for the first commercial TCP/IPv4 implementation.

UNIX and TCP/IP became very popular on college campuses, and with high end workstation vendors, such as Sun, Silicon Graphics and Intergraph. Personal computers were not powerful enough to run

UNIX until the Intel 386, at which point UCB ported the BSD version to the 386. However, as documented above, most personal computer networking was already moving to TCP/IPv4.

2.2.2 – Open System Interconnect (OSI)

While all of this was going on, ISO (the International Organization for Standardization) in Europe was creating a very thoroughly engineered suite of network protocols called Open System Interconnect (OSI), or more formally, X.200 (July 1994). This is where the famous "seven layer" network model comes from (TCP/IP is really based on a "four layer" model, which has caused no end of confusion among young network engineers). At one point the U.S. government decided to officially adopt OSI for its networking (this was called GOSIP, or Government Open Systems Interconnection Profile, defined in FIPS 146-1, 1990). Unfortunately, OSI was really more of an academic specification, not a real working network system, like TCP/IP was. After many years, GOSIP was finally abandoned and TCP/IPv4 was deployed, but GOSIP's legacy has hindered the adoption of IPv6 in the United States ("here we go again – GOSIP Part 2!"). X.400 E-mail and X.500 directory systems were built on top of OSI, and will not run on TCP/IP without substantial compatibility layers. One small part of X.500 (called X.509) was the source of digital certificates and Public Key Infrastructure, still used today. Lightweight Directory Access Protocol (LDAP) was an attempt to create an X.500-like directory system for TCP/IP based networks. That's about all that is left of the mighty OSI effort today, outside of Computer Science textbooks and Cisco Press books.

TCP/IPv6 was based heavily on TCP/IPv4, and was defined by the same group that defined TCP/IPv4. It is the only "natural" and straightforward evolutionary step after TCP/IPv4. I hope its detractors do not plan on staying with the first generation of TCP/IP forever, or try to move to something radically different from TCP/IPv4.

2.2.3 – E-mail Standardization

By this time, essentially all computer vendors had standardized on TCP/IP, but there were still a lot of competing standards for E-mail, including Microsoft's MS-Mail, Lotus's cc:Mail, and MCI Mail. The Internet folks used a much simpler E-mail standard called SMTP (Simple Mail Transfer Protocol). It first became the connecting backbone between various E-mail products (everyone had their E-mail to SMTP gateways, so users could exchange messages across organizations). Soon, everyone started using SMTP (together with POP3 and later IMAP) all the way to the end-user. Today virtually all E-mail worldwide is based on SMTP and TCP/IP.

2.2.4 – Evolution of the World Wide Web

Several other Internet applications evolved, including WAIS (Wide Area Information Server, for storing and retrieving documents) and Archie (the very first search engine). In turn, these efforts were merged with the idea of HyperText (documents with multi-level links) and evolved into HTML (HyperText Markup Language) and HTTP (HyperText Transfer Protocol). The World Wide Web was off and running. The first web browser and web server were created at the National Center for Supercomputing Applications (NCSA) at the University of Illinois, Urbana-Champaign campus. The people that created

those software projects (primarily Marc Andreessen and Eric Bina) were soon hired by Jim Clark, one of the founders of Silicon Graphics, to start NetScape, one of the most successful companies in the First Internet. They created a new and more powerful web server (NetScape Web Server) and web browser (NetScape Navigator). Interestingly enough, the original browser created by Andreessen at NCSA later became the starting point for Microsoft's Internet Explorer web browser.

2.3 – And That Brings Us Up to Today

That pretty much brings us up to the present day where the entire world has standardized on TCP/IPv4 protocols for both LANs (Local Area Networks) and WANs (Wide Area Networks). Multiprotocol Label Switching (MPLS) is not a competitor to TCP/IP, it is one more alternative at the Link Layer, peer to Ethernet and WiFi. More and more companies and organizations built TCP/IP networks and connected them together to create the Internet. Major telcos provided "backbone" WAN connections and dial-up access service (soon known as ISPs or Internet Service Providers). As the number of users (and the amount of traffic) on the Internet grew exponentially, Internet Exchange Points were created around the world. These are places where ISPs connect to each other so that traffic from a user of any provider can reach users of any other provider, worldwide.

Chapter 3 – Review of TCP/IPv4

This chapter is a brief review of TCP/IPv4, the foundation of the First Internet. Its purpose is to help you understand what is new and different in TCP/IPv6. It is not intended to be comprehensive. There are many great books listed in the bibliography if you wish to understand TCP/IPv4 at a deeper level. The reason it is relevant is because the design of IPv6 is based heavily on that of IPv4. First, IPv4 can be considered one of the great achievements in IT history, based on its worldwide success, so it was a good model to copy from. Second, there were several attempts to do a new design "from the ground up" with IPv6 (a "complete rewrite"). These involved *really* painful migration and interoperability issues. You need to understand what the strengths and weaknesses of IPv4 are to see why IPv6 evolved the way it did. You can think of IPv6 as "IPv4 on steroids", which takes into account the radical differences in the way we do networking today, and fixing problems that were encountered in the first 27 years of the Internet, as network bandwidth and number of nodes increased exponentially. We are doing things over networks today than *no one* could have foreseen a quarter of a century ago, no matter how visionary they were.

3.1 – Network Hardware

There are many types of hardware devices used to construct an Ethernet network running TCP/IP protocols. These include nodes, Network Interface Cards (NICs), cables, hubs, switches, routers and firewalls.

A *node* is a device (usually a computer) that can do processing and has some kind of wired or wireless connection(s) to a network. Examples of nodes are: desktop computers, notebook computers, netbooks, smart phones, hubs, switches, routers, wireless access points, network printers, network aware appliances, and so on. A node could be as simple as a temperature sensor, with no display and no keyboard, just a connection to a network. It could have a display and keyboard, or be a "headless node" with a management interface accessed via the network with Telnet, Secure Shell (SSH) or a web browser. All nodes connected to a network must have at least one valid IP address (per interface). If a node has only *one* network interface, such as a workstation computer, it is called a *host*. If a node has *multiple* interfaces connected to different networks, and the ability to forward packets between them, it is called a *gateway*. Routers and firewalls are special types of gateways that can forward packets between networks and or control traffic in various ways as it is forwarded. Gateways make it possible to build *internetworks*. They are described in more detail under *IPv4 Routing* in this chapter.

A *NIC* (or *Network Interface Card*) is the physical interface that connects a node to a network. It may also be called an *Ethernet adapter*. It should have a female RJ-45 connector on it (or possibly coax or fiber optic connector). It could be an actual add-in Peripheral Computer Interconnect (PCI) card. It could be integrated on the device's motherboard. It could also be a something that makes a wireless connection to a network, using Wi-Fi, WiMAX or similar standard. Typically all NICs have a globally unique, hard-wired MAC address (48 bits long, assigned by the manufacturer). A node can have one or more NICs (also called *interfaces*). Each interface can be assigned one or more IP addresses, and various other relevant network configuration items, such as the address of the default gateway and the addresses of the DNS servers.

Network cables today are typically *unshielded twisted pair* (UTP) cables that actually have *four* pairs of plastic coated wires, with each pair forming a twisted coil. They have RJ-45 male connectors on each end. They could also be fiber optic cables for very high speed or long run connections. Often today, professional contractors install UTP cables through the walls, and bring them together at a central location (sometimes called the *wiring closet*) where they are connected together to form a star network. Cables typically are limited to 100 meters or less in length, but the maximum acceptable length is a factor of several things, such as network speed and cable design. Modern cables rated as "CAT5" or "CAT5E" are good up to 100 Mbit/sec, while cables rated as "CAT6" are good up to a gigabit. Above that speed, you should be using fiber optic cables. It is also possible for twisted pair cables to be shielded if required to prevent interference from (or with) other devices.

A *network hub* is a device that connects multiple cables together so that any packet transmitted by any node connected to that hub is relayed to all the other nodes connected to the hub. It typically has a bunch of female RJ-45 connectors in parallel (called *ports*). In effect it ties together the network cables plugged into it into a star network. Hubs have a speed rating, based on what speed Ethernet they support. Older hubs might be only 10 Mbit/sec. More recent ones might be "Fast Ethernet", which means they support 100 Mbit/sec. If you have 5 nodes (A, B, C, D and E) connected together with a hub, and node B sends a packet to node D, all nodes, including A, C and E will see the traffic. The nodes not involved in the transaction will typically just discard the traffic. This dropping of packets not addressed to a node is often done by the hardware in the NIC, so that it never interrupts the software driver. Many NICs have the ability to be configured in *promiscuous mode*. When in this mode, they will accept packets (and make them available to any network application) whether those packets are addressed to this node or not. If this mode is selected, the dropping of packets not addressed to you must be done in software. However sometime you *want* to see all traffic on the subnet. For instance, this would be useful with Intrusion Detection, for diagnostic troubleshooting, or for collecting network statistics. Hubs come in various sizes, from 4 ports up to 48 ports, and can even be coupled with other hubs to make large network "backbones". You can also have a hierarchy of hubs, where several hubs distributed around a company actually connect in to a larger (and typically faster) central hub. Hubs do no processing of the packets, they are really just a cluster of repeaters that clean up and relay any incoming signals from any port to all the other ports. Actually, hubs are quite rare today, most such devices today are actually *switches*.

A *network switch* is similar to a network hub, but has some control logic in that minimizes unnecessary traffic. It partitions a LAN into multiple *collision domains* (one per switch port). Again, say you have a switch with cables connected to nodes A, B, C, D and E. If B sends a packet to D, that packet will be sent out *only* to the port to which D is connected. Switches *learn* what nodes are connected to what ports by maintaining a table of MAC addresses versus port number. When a switch is first powered on, this table is empty.

If node A (connected to port 1) sends a packet to node B (connected to port 2), the switch adds the MAC address of A, and the port it was seen on (1) to its table. In the future, when packets for A's MAC address come in any port, they will only be sent out port 1. Since the switch hasn't previously seen the MAC address of B (as a source address), it doesn't know where B is located, so it sends this first packet out *all* ports. If B replies to A's packet, the switch adds B's MAC address and port (2) to the table. In the future, packets sent to B's MAC address will only be sent out port 2. Each addition to the table expires after a certain amount of time, to allow nodes to be moved to other ports. An incoming packet sent to a

broadcast address will always be sent out *all* ports. This behavior holds down excessive traffic that would normally just be dropped anyway by the unaddressed nodes (not to mention unnecessary packet collisions). It also provides a small degree of privacy, even if someone enables their NIC in promiscuous mode. If your LAN is built using switches instead of hubs, you can typically only sniff traffic originating or terminating on the network segment connected to your port of the switch. Most switches are oblivious to IP addresses – they work only with MAC addresses. Because of this, they are *IP version agnostic*. This means they will carry IPv4 or IPv6 traffic (or even other kinds of Ethernet traffic) so long as that traffic uses Ethernet frames with MAC addresses.

If you are using a switch, but one of your connected nodes really *does* want to see traffic from other network segments, some switches have a *mirror port* function that will allow all traffic from any combination of ports to be copied to one port, to which you connect the node that wants to monitor that traffic. This must be configured, which requires a management interface of some kind. Like hubs, switches come in various speeds, from 10 Mbit/sec up to 1000 Mbit/sec (Gigabit). Unlike hubs, you can mix different speed nodes (10 Mbit/sec, 100 Mbit/sec and even 1000 Mbit/sec) on a single switch, so the speed rating it the *maximum* speed for nodes connected to it. Switches also come in sizes from 4 ports up to 48 ports, and better ones can be "stacked" (linked together) to effectively build a single giant switch. Lower end (cheaper) switches may have few if any configuration options, and may not even have a user interface. *Smart* (or *managed*) switches typically have a sophisticated GUI management interface (accessible via the network, usually over HTTP), or Command Line Interface (accessible either via serial port, telnet or SSH) that allows you to configure various things and/or monitor traffic. Switches also typically include support for monitoring or control using SNMP (Simple Network Monitoring Protocol). Very advanced switches have the ability to configure VLANs (Virtual Local Area Networks), which allow you to effectively create multiple sub-switches that are not *logically* connected together. Some of these advanced functions process IP addresses (*layer 3* functionality), hence are IP version *specific* (an IPv4-only smart switch cannot process IPv6 addresses, but the basic switch functionality may work fine). Very recent smart switches do support both IPv4 and IPv6 (dual stack), for layer 3 functionality.

3.2 – RFCs: The Internet Standards Process

Anyone studying the Internet, or developing applications for it, must understand the RFC system. RFC stands for *Request For Comments*. These are the documents that define the Internet Protocol Suite (the official name for TCP/IP) and many related topics. Anyone can submit an RFC. Ones that are part of the *Standards Track* are usually produced by **IETF** (Internet Engineering Task Force) *working groups*. Anyone can start or participate in a working group. Submitted RFCs begin life as a series of *Internet Drafts*, each of which has a lifespan of six months or less. Most drafts go through considerable peer review, and possibly quite a few revisions, before they are approved, are issued an official RFC number (e.g. 793) and become part of the official RFC collection. There are other kinds of documents in addition to the Standards Track, including information memos (FYI), humor (primarily ones issued on April 1) and even one obituary, for Jon Postel, the first RFC Editor and initial allocator of IP addresses, RFC 2468, "I Remember IANA", October 1998. There is even an RFC *about RFCs*, RFC 2026, "The Internet Standards Process, Revision 3", October 1996. That is a good place to start if you really want to learn how to read them.

The Internet standards process is quite different from the standards process of **ISO** (The International Standards Organization) that created the Open System Interconnect (OSI) network specification. ISO typically develops large, complex standards with multiple 4 year cycles, with hundreds of engineers and

much political wrangling. This was adequate for creating the standards for the worldwide telephony system, but is far too slow and hidebound for something as freewheeling and rapidly evolving as the Internet. The unique standards process of the IETF is one of the main reasons that TCP/IP is now the dominant networking standard worldwide. By the time OSI was specified, TCP/IP was already created, deployed, and being revised and expanded. OSI never knew what hit it.

Learning to read RFCs is an acquired skill, one that anyone serious about understanding the Internet, and most developers creating things for it, should master. There are certain "terms of art" (terms that have precise and very specific meanings), like the usage of MUST, SHOULD, MAY and NOT RECOMMENDED. As an example, the IPv6 Ready Silver (Phase 1) tests examine only the MUST (mandatory) items from relevant RFCs, but the IPv6 Ready Gold (Phase 2) tests also examine all of the SHOULD (optional) items.

RFCs are readily available to anyone for free. Compare this to the ISO standards, which can cost over $1000 for a complete set of "fascicles" for something like X.500. Today you can obtain RFCs easily in various formats by use of a search engine such as Google or Yahoo. The "official" source is the URL:

http://www.rfc-editor.org/rfc/rfcXXXX.txt (where XXXX is the RFC number)

There is also an official RFC search page, where you can search for phrases (like "TCP") in different tracks, such as RFC, STD, BCP or FYI, or all tracks. You can retrieve the ASCII or PDF versions. It is at:

https://www.rfc-editor.org/rfcsearch.html

There are over 5500 RFCs today. I have included many references to the relevant RFCs in this book. If you want to see all the gory details on any subject, go right to the source and read it. You may find it somewhat tough going until you learn to read "RFC-ese". A number of books on Internet technology are either just a collection of RFCs, or RFCs make up a large part of the content. There is no reason today to do that – anyone can download all the RFCs you want, and have them in soft (searchable) form. I have not included the text of even a single RFC in this book (warning – if you try to read this book somewhere without Internet access like on a plane, you may want to look ahead and download any relevant RFCs while you have Internet access). The casual reader should not need to reference the actual RFCs. The complete set of RFCs is probably tens of thousands of pages, and growing daily.

Most of the topics covered in this book also have considerable coverage on the Internet outside of the RFCs, such as in Wikipedia. Again, if you want to drill deeper in any of these topics, crank up your favorite search engine and have at it. The information is out there. What I've done is to try to collect together the essential information in a logical sequence, with a lot of explanations and examples, plus all the references you need to drill as deep as you like. I taught cryptography and Public Key Infrastructure for VeriSign for two years, so I have a lot of experience trying to explain complex technical concepts in ways that reasonably intelligent people can easily follow. Hopefully you will find my efforts worthwhile.

3.3 – TCP/IPv4

The software that made the First Internet (and virtually all Local Area Networks) possible has actually been around for quite some time. It is actually a suite (family) of protocols. The core protocols of this

suite are TCP (the *Transmission Control Protocol*) and IP (*Internet Protocol*), which gave it its common name, TCP/IP. Its official name is *The Internet Protocol Suite*.

TCP was first defined officially in RFC 675, "Specification of Internet Transmission Control Program", December 1974 (yes, 36 years ago). The protocol described in this document does not look much like the current TCP, and in fact, the Internet Protocol (IP) did not even exist at the time. Jon Postel was responsible for splitting the functionality described in RFC 675 into two separate protocols: (the new) TCP and IP. RFC 675 is largely of historical interest now. The modern version of TCP was defined in RFC 795, "Transmission Control Protocol – DARPA Internet Program Protocol Specification", September 1981 (seven years later). It was later updated by RFC 1122, "Requirements for Internet Hosts – Communication Layers" October 1989, which covers the link layer, IP layer and transport layer. It was also updated by RFC 3168, "The Addition of Explicit Congestion Notification (ECN) to IP", September 2001, which adds ECN to TCP and IP.

Both of these core protocols, and many others, will be covered in considerable detail in the rest of this chapter.

3.3.1 – Four Layer TCP/IPv4 Architectural Model

Unlike the OSI network stack, which really *does* have seven layers, the DoD network model has four layers, as shown below:

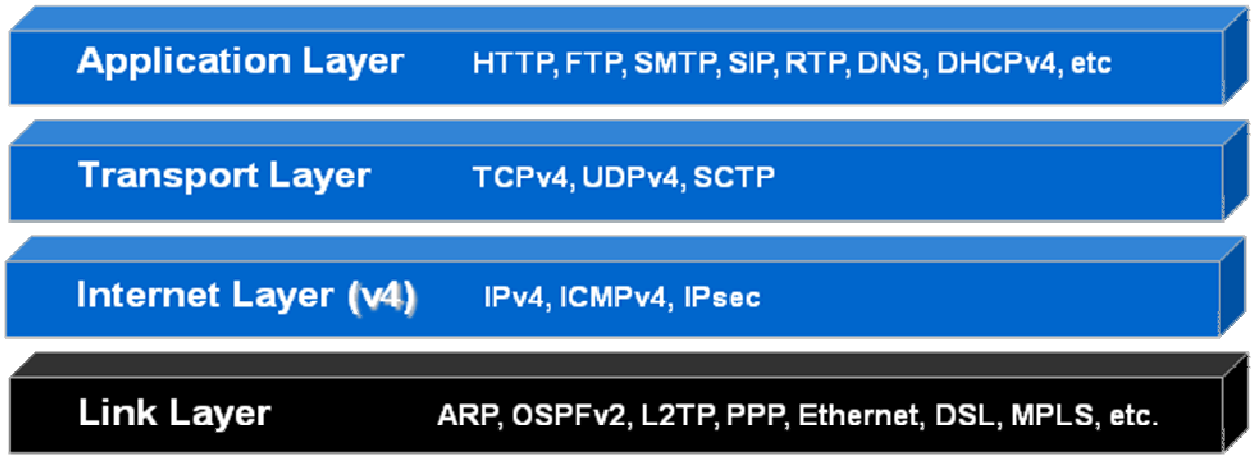


Figure 3.3-a: Four Layer TCP/IPv4 Model

It just confuses the issue to try to figure out which of the seven OSI layers the various layers of TCP/IP fit into. It is simply not applicable. It's like trying to figure out what color "sweet" is. The OSI seven layer model did not even exist when TCP/IP was defined. Unfortunately, many people use terms like "layer 2" switches versus "layer 3" switches. These refer to the OSI model. Books from Cisco Press and the Cisco certification exams are particularly adamant about using OSI terminology. Unfortunately hardly anyone actually uses OSI networks today. In this book we will try to consistently use the four layer model

terminology, while referring to the OSI terminology when necessary for you to relate the topic to actual products or other books.

Note: outgoing data begins in the application and is passed *down* the layers of the stack (adding headers at each layer) until it is written to the wire. Incoming data is read off the wire, and travels *up* the layers of the stack (having headers stripped off at each layer) until it is accepted by the application. In the following discussion, for simplicity, I describe only the outgoing direction.

The *Application Layer* implements the protocols most people are familiar with (e.g. HTTP). The software routines for these are typically contained in application programs such as browsers or web servers that make "system calls" to subroutines (or "functions" in C terminology) in the "socket API" (an API is an Application Program Interface, or a collection of related subroutines, typically supplied with the operating system or C programming language compiler). The application code creates outgoing data streams, and then calls routines in the API to actually send the data via TCP (Transmission Control Protocol) or UDP (User Datagram Protocol). Output to *Transport Layer*: *[DATA]* using IP addresses.

The *Transport Layer* implements TCP (the Transmission Control Protocol) and UDP (the User Datagram Protocol). These routines are internal to the Socket API. In the case of TCP, packet sequencing, plus error detection and retransmission are handled. This layer adds a TCP or UDP packet header to the data passed down from the *Application Layer*, and then pass the resulting packet down to the *Internet Layer* for further processing. Output to *Internet Layer*: *[TCP HDR[DATA]]*, using IP addresses.

The *Internet Layer* implements IP (the Internet Protocol) and various other related protocols such as ICMP (which includes the "ping" function among other things). The IP routine takes the data passed down from the *Transport Layer* routines, adds an IP packet header onto it, then passes the now complete IPv4 packet down to routines in the *Link Layer*. Output to Link layer: *[IP HDR[TCP HDR[DATA]]]* using IP addresses.

The *Link Layer* implements protocols such as ARP that convert IP addresses to MAC addresses. It also contains routines that actually read and write data (as fed down to it by routines in the *Internet Layer*) onto the network wire, in compliance with Ethernet or other standards. Output to wire: Ethernet *frame* containing the IP packet, using MAC addresses (or other Link Layer addresses for non-Ethernet networks).

Each layer "hides" the details (and/or hardware dependencies) from the higher layers. This is called "levels of abstraction". An architect thinks in terms of abstractions such as roofs, walls, windows, etc. The next layer down (the builder) thinks in terms of abstractions such as bricks, glass, mortar, etc. Below the level of the builder, an industrial chemist thinks in terms of formulations of clay or silicon dioxide to create bricks and glass. If the architect tried to think at the chemical or atomic level, it would be very difficult to design a house. His job is made possible by using levels of abstraction. Network programming is analogous. If application programmers had to think in terms of writing bits to the actual hardware, things like web browsers would be almost impossible. Each network layer is created by specialists that understand things at their level, and lower layers can be treated as "black boxes" by people working at higher layers.

Another important thing about network layers is that you can make major changes to one layer, without impacting the other layers much at all. The connections between layers are well defined, and don't

change (much). This provides a great deal of separation between the layers. In the case of IPv6, the Internet layer is almost completely redesigned internally, while the Link Layer and Transport Layer are not affected much at all (other than providing more bytes to store the larger IPv6 addresses). If your product is "IPv6 only", that's about the only change you would need to make to your application software (unless you display or allow entry of IP addresses). If your application is "dual stack" (can send and receive data over IPv4 or IPv6), then a few more changes are required in the application layer (e.g. to accept multiple IPv4 and IPv6 addresses from DNS and try connecting to one or more of them based on various factors, or to accept incoming connections over both IPv4 and IPv6). This makes it possible to migrate (or "port") network software (created for IPv4) to IPv6 or even dual stack with a fairly minor effort. In comparison, changing network code written for TCP/IP to use OSI instead would probably involve a complete redesign and major recoding effort.

3.3.2 – IPv4: The Internet Protocol, Version 4

IPv4 is the foundation of TCP/IPv4 and accounts for many of its distinguishing characteristics, such as its 32-bit address size, its addressing model, its packet header structure and routing. IPv4 was first defined in RFC 791 "Internet Protocol", September 1981.

The following standards are relevant to IPv4 (most important ones are in **bold**):

- **RFC 791, "Internet Protocol", September 1981 (Standards Track)**
- **RFC 792, "Internet Control Message Protocol", September 1981 (Standards Track)**
- **RFC 826, "An Ethernet Address Resolution Protocol", November 1982 (Standards Track)**
- RFC 1256, "ICMP Router Discovery Messages", September 1991 (Standards Track)
- RFC 2390, "Inverse Address Resolution Protocol", September 1998 (Standards Track)
- **RFC 2474, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", December 1998 (Standards Track)**
- RFC 4650, "HMAC-Authenticated Diffie-Hellman for Multimedia Internet KEYing (MIKEY)", September 2006 (Standards Track)
- RFC 4884, "Extended ICMP to Support Multi-Part Messages", April 2007 (Standards Track)
- RFC 4950, "ICMP Extensions for Multiprotocol Label Switching", August 2007 (Standards Track)
- RFC 5494, "IANA Allocation Guidelines for the Address Resolution Protocol (ARP)", April 1009 (Standards Track)
- **RFC 5735, "Special Use IPv4 Addresses", January 2010 (Best Current Practices)**

3.3.2.1 – IPv4 Packet Header Structure

So what are these packet headers mentioned above? In TCP/IPv4 packets, there is a TCP (or UDP) packet header, then an IPv4 packet header, then the packet data. Each header is a structured collection of data, including things such as the IPv4 address of the sending node, and the IPv4 address of the destination node. Why are we getting down to this level of detail? Because some of the big changes from IPv4 to IPv6 have to do with the new and improved IP packet header architecture in IPv6. In this chapter, we'll cover the IPv4 packet header. Here it is:

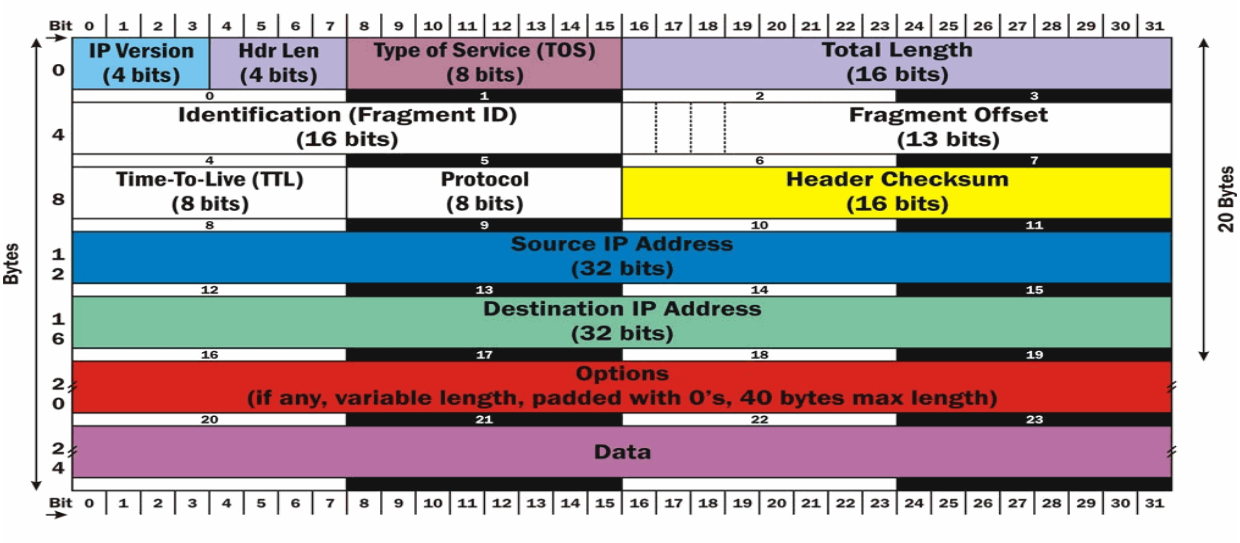


Figure 3.3-b: IPv4 Packet Header

The *IP Version* field (4 bits) contains the value 4, which in binary is "0100" (you'll never guess what goes in the first 4 bits of an IPv6 packet header!).

The *Header Length* field (4 bits) indicates how long the header is, in 32 bit "words". The minimum value is "5" which would be 160 bits, or 20 bytes. The maximum length is 15, which would be 480 bits, or 60 bytes. If you skip that number of words from the start of the packet, that is where the data starts (this is called the "offset" to the data). This will only ever be greater than 5 if there are options before the data part (which is not common).

The *Type of Service* field (8 bits) is defined in RFC 2474, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 headers", December 1998. This is used to implement a fairly simple QoS (Quality of Service). QoS involves management of bandwidth by protocol, by sender, or by recipient. For example, you might want to give your VoIP connections a higher priority than your video downloads, or the traffic from your boss higher priority than your co-worker's traffic. Without QoS, bandwidth is on a first-come-first served basis. 8 bits is not really enough to do a good job on QoS, and DiffServ is not widely implemented in current IPv4 networks. QoS is greatly improved in IPv6.

The *Total Length* field (16 bits) contains the total length of the packet, including the packet header, in bytes. The minimum length is 20 (20 bytes of header plus 0 bytes of data), and the maximum is 65,535 bytes (since only 16 bits are available to specify this). All network systems must handle packets of at least 576 bytes, but a more typical packet size is 1508 bytes. With IPv4, it is possible for some devices (like routers) to *fragment* packets (break them apart into multiple smaller packets) if required to get them through a part of the network that can't handle packets that big. Packets that are fragmented must be *reassembled* at the other end. Fragmentation and reassembly is one of the messy parts of IPv4 that got cleaned up a lot in IPv6.

The *Identification (Fragment ID)* field (16 bits) identifies which fragment of a once larger packet this one is, to help in reassembling the fragmented packet later. In IPv6 packet fragmentation is not done by intermediate nodes, so all the header fields related to fragmentation are no longer needed.

The next three bits are flags related to fragmentation. The first is reserved and must be zero (an April Fool's RFC once defined this as the "evil" bit). The next bit is the **DF** (Don't Fragment) flag. If DF is set, the packet cannot be fragmented (so if such a packet reaches a part of the network that can't handle one that big, that packet is dropped). The third bit is the **MF** (More Fragments) flag. If MF is set, there are more fragments to come. Unfragmented packets of course have the MF flag set to zero.

The *Fragment Offset* field (13 bits) is used in reassembly of fragmented packets. It is measured in 8 byte blocks. The first fragment of a set has an offset of 0. If you had a 2500 byte packet, and were fragmenting it into chunks of 1020 bytes, you would have three fragments as follows:

Fragment ID	MF Flag	Total Length	Data Size	Offset
1	1	1020	1000	0
2	1	1020	1000	125
3	0	520	500	250

The *Time To Live (TTL)* field (8 bits) is to prevent packets from being shuttled around indefinitely on a network. It was originally intended to be lifetime in seconds, but it has come to be implemented as "hop count". This means that every time a packet crosses a switch or router, the hop count is decremented by one. If it reaches zero, the packet is dropped. Typically if this happens, an ICMPv4 message ("time exceeded") is returned to the packet sender. This mechanism is how the *traceroute* command works. Its primary purpose is to prevent *looping* (packets running around in circles).

The *Protocol* field (8 bits) defines the type of data found in the data portion of the packet. Protocol numbers are not to be confused with *ports*. Some common protocol numbers are:

1	ICMP	Internet Control Message Protocol (RFC 792)
6	TCP	Transmission Control Protocol (RFC 793)
17	UDP	User Datagram Protocol (RFC 768)
41	IPv6	IPv6 tunneled over IPv4 (RFC 2473)
83	VINES	Banyan Vines IP
89	OSPF	Open Shortest Path First (RFC 1583)
132	SCTP	Streams Control Transmission Protocol

The *Header Checksum* field (16 bits). The 16-bit one's complement of the one's complement sum of all 16 bit words in the header. When computing, the checksum field itself is taken as zero. To validate the checksum, add all 16 bit words in the header together including the transmitted checksum. The result should be 0. If you get any other value, then at least one bit in the packet was corrupted. There are certain multiple bit errors that can cancel out, and hence bad packets can go undetected. Note that since the hop count (TTL) is decremented by one on each hop, the IP Header checksum must be recalculated at each hop. The IP Header Checksum was eliminated in IPv6.

The *Source Address* field (32 bits) contains the IPv4 address of the sender (may be modified by NAT).

The *Destination Address* field (32 bits) contains the IPv4 address of the recipient (may be modified by NAT in a reply packet).

Options (0 to 40 bytes) Not often used. These are not relevant to this book. If you want the details, read the RFCs.

Data – (variable number of bytes) The data part of the packet – not really part of the header. Not included in the IP Header checksum. The number of bytes in the data field is the value of 'Total Length', minus the value of 'Header Length'.

3.3.2.2 – IPv4 Addressing Model

In IPv4, addresses are 32 bits in length. They are simply numbers from 0 to 4,294,967,295. For the convenience of humans, these numbers are typically represented in *dotted decimal notation*. This splits the 32 bit addresses into four 8-bit fields, and then represents each 8-bit field with a decimal number from 0 to 255. These decimal numbers cover all possible 8 bit binary patterns from 0000 0000 to 1111 1111. The decimal numbers are separated by "dots" (periods). Leading zeros can be eliminated. The following are all valid IPv4 addresses represented in dotted decimal:

123.45.67.89	A globally routable address (valid anywhere on the Internet)
10.3.1.51	A private address (valid only within a single LAN, as per RFC 1918)
255.255.255.255	The broadcast address for IPv4
127.0.0.1	The loopback address for IPv4

Originally there were 5 *classes* of IPv4 addresses, as defined in RFC 791, "Internet Protocol", September 1981.

Class A – First bit 0 (0.0.0.0 – 127.255.255.255). 8 bit network number, 24 bit node within network number, subnet mask 255.0.0.0. There are 128 Class A networks, each containing 16.8M addresses.

Class B – First two bits '10' (128.0.0.0 – 191.255.255.255), 16 bit network number, 16 bit node within network number, subnet mask 255.255.0.0. There are 16,384 class B networks, each containing 65,536 addresses.

Class C – First three bits '110' (192.0.0.0 – 223.255.255.255), 24 bit network number, 8 bit node within network number, subnet mask 255.255.255.0. There are 2M class C networks, each containing 256 addresses.

Class D – First four bits '1110' (224.0.0.0 – 239.255.255.255), used for multicast.

Class E – First four bits '1111' (240.0.0.0 – 255.255.255.255), experimental / reserved (not passed by most routers).

Network Ports

Each IP address on a network node has 65,536 ports associated with it (the port number is a 16 bit value, and 2 to the 16^{th} is 65,536). Any of those ports can either be used to make an outgoing connection, or to accept incoming connections. There is a list of *Well Known Ports* which associates particular ports with certain protocols. For example, port 25 is associated with SMTP. There is nothing

magical (or E-mail-ish) about port 25. SMTP will work just as well on any other port, e.g. 10025. Use of port 25 for SMTP is simply a convention that many people adopt. Such conventions make it easier to locate the SMTP server on a node you might not be familiar with. To be specific, ports are a Transport Layer thing, and there are really 65,536 TCP ports, and another 65,536 UDP ports for each address. ICMP, which is an Internet Layer thing, does not have any port(s) associated with it.

When you deploy an Internet server (e.g. an SMTP server for sending and receiving E-mail), the software opens a *socket* (a programming abstraction) in *listen mode* on a particular port (in the case of SMTP, port 25). An E-mail client that wants to connect to it creates its own socket in *connect mode*, and tells it to connect to a particular IP address (that of the SMTP server) using a particular port (in this case 25). When the connection attempt reaches the server, the server detects the attempt, and accepts the connection (actually the port on the server that the connection is accepted on will be any available port, typically higher than 1024). A well written server would then make a clone of itself (this is called *forking* in UNIX-speak), then go back to listening for further connections, while its clone went ahead and processed the connection. When the processing is complete on a given connection, the sockets used would be closed (on both server and client), and the clone of the server will quietly commit suicide. In theory you could have thousands of clones of the server all simultaneously handling E-mail connections on a single server (given sufficient memory and other resources). Busy web servers (like those at Google) often have *many* thousands of connections being processed at any given time (but never more than 65,000 on a given interface – each connection uses up one port).

In UNIX, ports with numbers under 1024 are *special*, and only software that has root privilege can use them. Most common Internet services use ports in that range. There are many *Well Known Ports*, but here are a few of the more common ones:

22 – SSH (Secure Shell)
25 – SMTP (client to server and server to server E-mail protocol)
53 – DNS (Domain Name System)
80 – HTTP (world wide web)
110 – POP (server to client E-mail retrieval)
143 – IMAP (more modern server to client E-mail retrieval)
389 – LDAP (directory service)
443 – HTTPS (world wide web over SSL)

3.3.2.3 – IPv4 Subnetting

This leads us naturally into the topic of IPv4 subnetting. This is one of the more difficult areas of networking for people learning to work with IPv4. All addresses have two “parts”, the first part being the address of the network (e.g. 192.168.0.0), and the second being the node within that network (e.g. 0.0.2.5). These two parts can be split apart along some “bit boundary”. In this case, the address of the network is in the first 16 bits, and the node within network is in the last 16 bits. The addresses of all nodes in such a network share the same first 16 bits, but each has a unique last 16 bits. So, such a network might have nodes with addresses 192.168.2.5, 192.168.3.7 and 192.168.200.12, but *not* one with the address 192.169.2.1 (that address is in network *192.169.0.0*, not network 192.168.0.0).

A *subnet mask* is a 32 bit value in which the first n bits (n=1 to 32) have the value 1, and the remaining 32-n bits have the value 0. It is used to split an IPv4 address into its two parts (the first n bits, and the last 32-n bits). In the network just described, the subnet mask is *255.255.0.0* (the first 16 bits have the value 1, the last 16 bits have the value 0). You do a Boolean "AND" function of the address with the subnet mask to get the address of the network, and a Boolean "AND" of the address with the *one's complement* of the subnet mask (in this case 0.0.255.255) to get the node within subnet. This is difficult to visualize in dotted decimal. It is rather more obvious in binary. The Boolean "AND" function produces a 1 if both inputs are 1, else it produces a 0. The "one's complement" (Boolean "NOT") function changes each 0 to a 1, and each 1 to a 0. With the "AND" function, where there is a 1 in the mask, the corresponding bit of the address "flows through" to the result. Where there is a 0 in the mask, the corresponding bit of the address is blocked (forced to the value 0). The following example (with addresses and mask shown in both dotted decimal and binary) should make this clear:

```
Address              192.168.2.5         1100 0000 1010 1000 0000 0010 0000 0101
Subnet mask          255.255.0.0         1111 1111 1111 1111 0000 0000 0000 0000
AND                                      ---- ---- ---- ---- ---- ---- ---- ----
Network address      192.168.0.0         1100 0000 1010 1000 0000 0000 0000 0000

Address              192.168.2.5         1100 0000 1010 1000 0000 0010 0000 0101
NOT mask             0.0.255.255         0000 0000 0000 0000 1111 1111 1111 1111
AND                                      ---- ---- ---- ---- ---- ---- ---- ----
Node within network  0.0.2.5             0000 0000 0000 0000 0000 0010 0000 0101
```

For subnet mask 255.0.0.0 (Class A), the first 8 bits are network address, the last 24 are node within subnet.

For subnet mask 255.255.0.0 (Class B), the first 16 bits are network address, the last 16 are node within subnet.

For subnet mask 255.255.255.0 (Class C), the first 24 bits are network address, the last 8 are node within subnet.

Subnetting was easy when the three IP address classes (A, B & C) were used. The first few bits of the address determined the subnet mask. If the first bit of the 32 bit address was “0”, then the address was Class A, and the subnet mask was 255.0.0.0. If the first two bits of the address were “10”, then the address was class B, and the subnet mask for 255.255.0.0. If the first three bits of the address were “110”, then the address was class C, and the subnet mask was 255.255.255.0. This could actually be done automatically, so no one worried about subnet masks.

One of the changes made in the IPv4 addressing model in the mid 1990’s was to introduce Classless Inter-Domain Routing, in RFC 1519, “Classless Inter-domain Routing (CIDR)”, September 1993. It was later replaced by RFC 4632 “Classless Inter-domain Routing (CIDR)”, August 2006.

When CIDR was introduced, there were two consequences. First, the split between the two parts of the address could come at any bit boundary, not just after 8, 16 and 24 bits. Second, several small blocks (e.g. /28 blocks) could be carved out of a bigger block anywhere in the address space (perhaps from an old Class A block, such as 7.x.x.x), so you could no longer determine the correct subnet mask by looking at the first few bits of an address. For example, a “/8” (Class A) block might be carved up into 65,536 “/24” (Class C) subnets, which could be allocated to different organizations.

Let’s say your ISP, instead of giving you a Class C block, only gives you a “/28” block of real (routable) IPv4 addresses, which would be 16 real IPv4 addresses, e.g. 123.45.67.0 through 123.45.67.15. First, two of these addresses are not usable (may not be assigned to nodes). 123.45.67.0 is the “network address”, and 123.45.67.15 is the “broadcast address”. That leaves 14 usable addresses (123.45.67.1 through 12.45.67.14). So what is your subnet mask? If you check the above table of useful CIDR block sizes, a /28 subnet has a subnet mask of 255.255.255.240. In binary that is 1111 1111 1111 1111 1111 1111 1111 0000 (first 28 bits are 1, last 4 bits are 0). However, by the old rules (first bit is a 0) these are really from a “Class A” block, so the automatically generated subnet mask would have been 255.255.255.0, which is not correct.

Now, what if your organization really has 100 nodes that need IP addresses? How do you give each of them unique addresses if you only have 14 usable addresses to work with? That's where Network Address Translation (NAT) comes in (covered in the next chapter). If you think CIDR made your life more "interesting", wait until you see what NAT does to it! Getting rid of CIDR and NAT are one of the big wins in IPv6. In fact, you will find that the entire subject of subnets has become totally trivial.

3.3.2.4 – MAC Addresses

IPv4 addresses are not actually used at the lowest layer of the TCP/IPv4 network stack (the *Link Layer*). Each network hardware interface actually has a 48 bit "MAC address" burned into it by the manufacturer. The first 24 bits of this (called the "Organizationally Unique Identifier" or OUI) specify the manufacturer. The last 24 bits of this (called the "Network Interface Controller Specific" part) is assigned by each manufacturer, to be unique within a given OUI. This means that the entire 48 bit value is globally unique. For example, Dell Computer has a number of OUIs assigned to them by the IEEE, including 00-06-5B, 00-08-74 and 00-18-8B. If you encounter a NIC with a MAC address that has one of those sets of 24 bits, it was made by Dell Computer. When you use the command "ipconfig /all" in Windows, you get a list of network configuration information for all of your interfaces (some of which are "virtual"). If you look for "Local Area Connection", that is information about your main (or only) network connection to your LAN. Under that, you will see an item labeled "Physical Connection", followed by 6 pairs of hex digits, separated by dashes. That is the MAC address of your Network Interface Card (NIC). Mine is 00-18-8B-78-DA-1A. This means my NIC was made by Dell (my whole computer was, but the MAC address doesn't tell you that). Actually, since the NIC I'm using is on the motherboard (not an add-on PCI card), this does tell me the motherboard was made by Dell.

Network switches come in two varieties. "Layer 2" switches (which I would call *Link Layer* switches) only work with MAC addresses. They don't even "see" IP addresses. Hence "Layer 2" switches are IP version agnostic; they work equally well with IPv4 or IPv6, or a mixture of the two (dual stack). "Layer 3" switches (sometimes called "smart" switches) work with MAC addresses, but they *also* understand and can see IP addresses (these work at *both* the *Link Layer* and the *Internet Layer*, in TCP/IP-speak). They can do things like create VLANs (Virtual LANs) to segregate traffic based on IP addresses. An IPv4-only "layer 3 switch" cannot work with IPv6 traffic (or at least none of its "higher level" functions will affect IPv6 traffic). There are now a few dual-stack "layer 3" switches on the market, such as the SMC 8848M, which I happen to be running in my home network. I can even manage it over IPv6 (via web and SNMP), and create VLANs based on IPv6 addresses.

Mapping From IPv4 Addresses to MAC Addresses

The software in the *Application Layer*, the *Transport Layer*, and the *Internet Layer* of the TCP/IPv4 stack think in terms of IPv4 addresses. But the *Link Layer* (and the hardware) thinks in terms of MAC addresses. How do IPv4 addresses get mapped onto MAC addresses?

Address Resolution Protocol (ARP)

There are two protocols in TCP/IPv4 (that don't even exist in IPv6) called ARP (Address Resolution Protocol) and InARP (Inverse Address Resolution Protocol). These protocols live in the Link Layer. ARP

maps IP addresses onto MAC addresses. This is kind of like the mapping between FQDNs and IP addresses done in the Application Layer by DNS, but down in the Link Layer.

ARP is defined in RFC 826, "An Ethernet Address Resolution Protocol", November 1982. ARP operates only within the network segment (routing domain) that a host is connected to. It does not cross routers. It is used to determine the necessary MAC addresses to get a packet from one node in a subnet to another node in the same subnet (which could be a "default gateway" node that knows how to relay it on further). But for the hop from the sender to the default gateway, it is the same problem as getting the packet to any other local node. When the sender goes to send a packet, if the recipient's address is on the local link, an ARP request is done for the recipient's address, and the packet is sent to the recipient. If the recipient's address is *not* on the local link, an ARP request is done instead for the sender's default gateway address, and the packet is sent to the default gateway node, which will then worry about forwarding it on towards the real recipient.

Say Alice (one TCP/IPv4 node) wants to send a packet to Bob (another TCP/IPv4 node, on the same network segment). Assume Alice does not currently know Bob's MAC address. Each machine has a table of IP addresses and MAC addresses (called an *ARP table*). At this time, there is no entry in Alice's ARP table with Bob's IP address and MAC address. So, Alice *first* sends an Ethernet ARP request to all machines on the network segment (using the Ethernet broadcast address), with the following info:

> opcode = REQUEST
> hardware type = Ethernet,
> protocol = IPv4
> sender's IP address = Alice's IPv4 address (she knows her own IP address)
> sender's MAC address = Alice's MAC address (she knows her own MAC address)
> recipient's IP address = Bob's IPv4 address (she knows Bob's IP address)
> recipient's MAC address = "don't care", all zeros (she doesn't know Bob's MAC address yet)

All machines on the network segment will receive the packet, but everyone other than Bob will ignore it ("not for me – IGNORE!"). Bob understands Ethernet protocol and IPv4 protocol. He recognizes his own IPv4 address ("it's for ME!"). He adds Alice's IPv4 address and Alice's MAC address into his ARP table (for future reference), then sends a response Ethernet ARP packet back to Alice, using her MAC address (which he now knows) instead of the broadcast address, with the following info:

> opcode = RESPONSE
> hardware type = Ethernet
> protocol = IPv4
> sender's IP address = Bob's IPv4 address (he knows his own IP address)
> sender's MAC address = Bob's MAC address (he knows his own MAC address)
> recipient's IP address = Alice's IPv4 address (from the request)
> recipient's MAC address = Alice's MAC address (from the request)

Only Alice gets the response (this was not a broadcast). Alice sees that this is a RESPONSE, and the sender's address tells her who the response was from. Alice then adds Bob's IP address and MAC address into her ARP table. Now that she knows how to send things to Bob, she goes ahead with sending the packet that she originally was trying to send. This process is called *Address Resolution*. Hence the name *Address Resolution Protocol*.

The ARP table has expiration times (TTL), and when an entry becomes "stale" it will be discarded, and the next time a packet is sent to that address, a new fresh entry will be added to the ARP table.

In Windows, you can view your ARP table at any time, in a DOS Window, with the command "arp –a". The results might look something like this:

```
C:\Users\lhughes.HUGHESNET>arp -a

Interface: 172.20.2.1 --- 0xb
  Internet Address      Physical Address      Type
  172.20.0.1            00-1b-21-1d-c1-59     dynamic
  172.20.0.11           00-17-a4-ec-11-9c     dynamic
  172.20.0.12           00-e0-81-47-fa-ce     dynamic
  172.20.0.13           00-15-f2-2e-b4-1c     dynamic
  172.20.0.21           00-18-f3-2e-32-87     dynamic
  172.20.0.88           00-14-fd-12-fa-5a     dynamic
  172.20.1.6            00-1e-90-1e-5b-4f     dynamic
  172.20.1.8            00-1e-65-97-de-e0     dynamic
  172.20.1.9            00-15-f2-2e-b4-1c     dynamic
  172.20.255.255        ff-ff-ff-ff-ff-ff     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.252           01-00-5e-00-00-fc     static
  224.111.140.122       01-00-5e-6f-8c-7a     static
  226.196.145.70        01-00-5e-44-91-46     static
  237.62.223.84         01-00-5e-3e-df-54     static
  239.255.255.250       01-00-5e-7f-ff-fa     static
```

Inverse ARP (InARP)

There is another protocol called Inverse ARP (InARP) that maps MAC addresses onto IP addresses. This is needed only by a few network hardware devices (like ATM). It works almost exactly like ARP, except different opcodes are used and the sender sends the recipient's MAC address (which it knows) but zero fills the recipient's IP address (which it wants to know). The recipient recognizes its own MAC address and responds with the same information that it does to an ARP. The older RARP (Reverse ARP) protocol is now deprecated.

3.3.3 – Types of IPv4 Packet Transmissions

The most common type of packet transmission is *unicast*. This is when one node (A) sends a packet to just one other node (B). A and B can be in the same local link, or halfway around the world, so long as routable IP addresses are used, and a routing path is available between A and B. It is still called unicast.

Another kind of transmission is *broadcast* (covered in more detail below). Here a node can transmit a packet to *all* nodes in the local link. Usually any node not interested in a broadcast packet will just drop it. If the packet was an ICMP echo request (ping), all nodes on the local link might reply to it, which could cause a lot of excess traffic.

There is another kind of transmission called *anycast*. Here a node can transmit a packet to a single node out of a set of some collection of nodes (e.g. the "nearest" DNS server). Usually only a single node will accept the transmission and reply to the sender. This mechanism is somewhat limited in IPv4, but works really well in IPv6. DNS anycast is used with the root DNS servers to allow multiple copies of each root server, to handle the load and minimize turnaround on root server requests.

There is one more kind of transmission called *multicast*. Here one node can send a single stream of packets, such as a digitized radio program, and any number of recipient nodes can *subscribe* to that multicast and receive it. Usually listening is a passive act, no responses are sent to the sender. Typically the sender has no knowledge of which, or even how many nodes receiving the transmission. It is efficient because other nodes further along the network handle replication of the traffic to nodes beyond them. This is analogous to many radios receiving a transmission from a single radio transmitting station. This is covered in more detail below. This is supported in IPv4, but works far better in IPv6.

3.3.3.1 – IPv4 Broadcast

Any node can send a packet to a special IPv4 address (255.255.255.255) and all nodes on the local link will receive it. Usually, there is some kind of information in the packet that allows most nodes to realize that packet does not concern them (e.g. if broadcast packet contains a DHCPv4 message, all nodes that don't have a DHCPv4 server will ignore it). This mechanism can help locate servers or solve other problems (like not yet having a valid IP address), but it can put unnecessary loads on all nodes that aren't involved. It can also lead to broadcast storms, which involve massive amounts of useless traffic clogging or totally shutting down an IPv4 network. As an example, a "smurf attack" sends zillions of pings to the broadcast address with the request containing the *spoofed* address of the node under attack as the "source address" (who sent the message). All nodes on the local link "respond" to the poor node under attack, which *amplifies* the attack. There are certain kinds of misconfigurations or hardware failures in network switches that can cause broadcast storms as well.

Packets sent to the broadcast address do not cross routers (or VLAN boundaries), so appropriate use of these can limit the extent of disruption due to excessive broadcasts or storms. The set of nodes that a broadcast will reach is called a *broadcast domain*.

Broadcast is used in the DHCPv4 protocol, to allow a node to find and communicate with the DHCPv4 server, before it even gets an address.

Broadcast does not exist in IPv6, because it can be so trouble prone. Other mechanisms are used to locate DHCPv6 servers, or solve other problems for which broadcast may be used in IPv4.

3.3.3.2 – IPv4 Multicast

Multicast allows a node to transmit a stream of data (usually as UDP datagrams) to one of a number of special "multicast" addresses. Any number of other nodes can subscribe to that address and receive the datagrams. As one example, this could be used to send "broadcast" (in the media sense) radio or television programs.

Sites like YouTube, and services like “on-demand” television, use traditional unicast (one sender connecting to one recipient) transmissions to each user. This requires a great deal of bandwidth, and a powerful network infrastructure at the transmission site, especially if there are a large number of recipients (potentially millions). Multicast is necessary to bring costs and network bandwidth requirements low enough to make it competitive with media broadcast over satellite or cable systems.

There are several mechanisms and protocols involved in IPv4 multicast:

- An IP multicast group address (one of the IPv4 “Class D” addresses described above)
- A sending node which can convert some kind of data such as audio and/or video into digital form, and transmit the resulting UDP packets to that multicast group address
- A multicast distribution tree, where every router crossed supports multicast operation
- A new protocol called IGMP (Internet Group Management Protocol) that allows clients to subscribe to a particular multicast transmission
- Another new protocol called PIM (Protocol Independent Multicast) that sets up multicast distribution trees
- Clients that can “subscribe” to specific multicast addresses (receive the data being transmitted by the sender) and process the received digital data into some kind of service, such as audio or video

Assuming there is a multicast program available on a particular multicast address (e.g. 239.1.2.3) a consumer can use a multicast client application to extend the distribution tree associated with that address to reach to his computer. This corresponds to selecting a channel on a television. There may be multiple routers between the sender and this subscriber. All of those routers must support multicast, and be informed to replicate packets from that sender to that recipient. IGMP is used to subscribe to a specific multicast address, and PIM is used to inform all intervening routers to extend the distribution tree to this client. The multicast server does not need to know anything about the recipients, and normally does get any response from them. The creation of the distribution tree and subscriptions to particular multicast addresses are handled by the clients and intervening multicast routers, not by the multicast server.

Unlike unicast routers, a multicast router does not need to know how to reach all possible distribution trees, only those for which it is passing traffic from a sender to a recipient. If there is no recipient subscribed to a given channel “downstream” from a router (from the sender to recipient), there is no need for it to replicate packets and forward them downstream. If a recipient downstream from that router subscribes to a particular address, then that router will start replicating incoming upstream packets from the multicast address and relay them downstream towards that recipient (or recipients). This is called adding a “graft” onto the tree. If there are recipients downstream on a particular path from a multicast router, and the last one “tunes out”, then that router is informed to stop replicating packets along that path. This is called a “prune” of the distribution tree. It is possible that one subscriber “tuning out” could result in an entire chain of multicast routers being pruned, if there are no other subscribers on that subtree.

Multicast is often used for services such as IPTV, including applications such as distance learning. Not all IPv4 routers support multicast and the related protocols, so IPv4 multicast works best in “walled garden” networks, for example within a single ISP’s network (e.g. Comcast subscriber accessing

multicast content from Comcast). In such a situation it is possible to insure that all intervening routers support the necessary protocols (which are optional in IPv4).

It is possible to build a fully IPv4 multicast compliant router using open source operating systems and an open source package called XORP (eXtensible Open Router Platform, at *www.xorp.org*). XORP was first developed for FreeBSD, but is available on Linux, OpenBSD, NetBSD and Mac OS X. The XORP technology and team has recently been transferred to a commercial startup backed by VCs (called XORP Inc). Many modern enterprise-class routers support IPv4 multicast, but not all do. Not many Small Office / Home Office (SOHO)-class routers do. In IPv6, Multicast is an integral part of the standard, and support is mandatory in all compliant devices. It also works in a very different way, and is much more scalable.

Internet Relay Chat (IRC) uses a different approach to multicast (not the standard multicast protocols), and creates a spanning tree across its overlay network to all nodes that subscribe to a given chat channel. Unlike multicast delivered media content, IRC is a two-way channel.

Standards relevant to IPv4 multicast include:

- **RFC 1112, "Host Extensions for IP multicasting", August 1989 (Standards Track)**
- RFC 2236, "Internet Group Management Protocol, Version 2", November 1997 (Standards Track)
- **RFC 2588, "IP Multicast and Firewalls", May 1999 (Informational)**
- **RFC 2908, "The Internet Multicast Address Allocation Architecture", September 2000 (Informational)**
- **RFC 3376, "Internet Group Management Protocol, Version 3", October 2002 (Standards Track)**
- RFC 3559, "Multicast Address Allocation MIB", June 2003 (Standards Track)
- RFC 3973, "Protocol Independent Multicast – Dense Mode (PIM-DM)", January 2005 (Experimental)
- RFC 4286, "Multicast Router Discovery", December 2005 (Standards Track)
- RFC 4541, "Considerations for Internet Group Management Protocol (IGMP) and Multicast Listener Discovery Protocol (MLD) Snooping Switches", May 2006 (Informational)
- **RFC 4601, "Protocol Independent Multicast – Sparse Mode (PIM-SM): Protocol Specification (Revised)", August 2006 (Standards Track)**
- RFC 4604, "Using Internet Group Management Protocol Version 3 (IGMPv3) and Multicast Listener Discovery Protocol Version 2 (MLDv2) for Source-Specific Multicast", August 2006 (Standards Track)
- RFC 4605, "Internet Group Management Protocol (IGMP) / Multicast Listener Discovery (MLD)-Based Multicast Forwarding (IGMP/MLD Proxying)", August 2006 (Standards Track)
- RFC 4607, "Source Specific Multicast for IP", August 2006 (Standards Track)
- RFC 4610, "Anycast-RP Using Protocol Independent Multicast (PIM)", August 2006 (Standards Track)
- **RFC 5015, "Bidirectional Protocol Independent Multicast (BIDIR-PIM)", October 2007 (Standards Track)**
- RFC 5060, "Protocol Independent Multicast MIB", January 2008 (Standards Track)
- **RFC 5110, "Overview of the Internet Multicast Routing Architecture", January 2008 (Informational)**
- RFC 5135, "IP Multicast Requirements for a Network Address Translator (NAT) and a Network Address Port Translator (NAPT)", February 2008 (Best Current Practices)

- RFC 5332, “MPLS Multicast Encapsulations”, August 2008 (Standards Track)
- RFC 5374, “Multicast Extensions to the Security Architecture for the Internet Protocol”, November 2008 (Standards Track)
- RFC 5384, “The Protocol Independent Multicast (PIM) Join Attribute Format”, November 2008 (Standards Track)
- RFC 5401, “Multicast Negative-Acknowledgement (NACK) Building Blocks”, November 2008 (Standards Track)
- RFC 5519, “Multicast Group Membership Discovery MIB”, April 2009 (Standards Track)
- RFC 5740, “NACK-Oriented Reliable Multicast (NORM) Transport Protocol”, November 2009 (Standards Track)
- **RFC 5771, “IANA Guidelines for IPv4 Multicast Address Assignments”, March 2010 (Best Current Practice)**
- RFC 5790, “Lightweight Internet Group Management Protocol Version 3 (IGMPv3) and Multicast Listener Discovery Version 2 (MLDv2) Protocols”, February 2010 (Standards Track)

3.3.3.3 – Internet Group Management Protocol (IGMP)

IGMP is an *Internet Layer* protocol that supports IPv4 multicast. It manages the membership of IPv4 multicast groups, and is used by network hosts and adjacent multicast routers to establish multicast group membership. There are three versions of it so far. IGMPv1 is defined in RFC 1112, “Host Extensions for IP Multicasting”, August 1989. IGMPv2 is defined in RFC 2236, “Internet Group Management Protocol, Version 2”, November 1997. IGMPv3 is defined in RFC 3376, “Internet Group Management Protocol, Version 3”, October 2002.

Some “Layer 2” switches have a feature called “IGMP snooping”, which allows them to look at the “Layer 3” packet content, to enable multicast traffic to go only to those ports which have subscribers on them, while blocking it (and thereby reducing unnecessary traffic) on ports with no subscribers. A switch without IGMP snooping will flood all connected nodes in the broadcast domain with all multicast traffic. This can be used by hackers to “deny service” to clients that are too busy receiving and ignoring multicast traffic to handle useful traffic. This is called a Denial of Service, or DoS attack. Active IGMP snooping is described in RFC 4541, “Considerations for Internet Group Management Protocol (IGMP) and Multicast Listener Discovery Protocol (MLD) Snooping Switches”, May 2006.

3.3.3.4 – Protocol Independent Multicast (PIM)

PIM supports IPv4 multicast. It is called “protocol independent” because it does not include its own network topology discovery mechanism. PIM does not include routing, but provides multicast forwarding by using static IPv4 routes, or routing tables created by IPv4 routing protocols, such as RIP, RIPv2, OSPF, IS-IS or BGP.

PIM Dense Mode is defined in RFC 3973, “Protocol Independent Multicast – Dense Mode (PIM-DM)”, January 2005. This uses dense multicast routing, which builds shortest-path trees by flooding multicast traffic domain wide, then pruning branches where no receivers are present. It does not scale well.

PIM Sparse Mode is defined in RFC 4601, "Protocol Independent Multicast – Sparse Mode (PIM-SM)", August 2006. PIM-SM builds unidirectional shared trees routed at a rendezvous point per group, and can create shortest-path trees per source. It scales fairly well for wide-area use.

Bidirectional PIM is defined in RFC 5015, "Bidirectional Protocol Independent Multicast (BIDIR-PIM)", October 2007. It builds shared bi-directional trees. It never builds a shortest-path tree, so there may be longer end-to-end delays, but it scales very well.

3.3.4 – ICMPv4: Internet Control Message Protocol for IPv4

ICMPv4 is a key protocol in the *Internet Layer* that complements version 4 of the Internet Protocol (IPv4). It was originally defined in RFC 792, "Internet Control Message Protocol", September 1981. There are a number of ICMP messages defined. Some of these are generated by the network stack in response to errors in datagram delivery. Some are used for diagnostic purposes (to check for network connectivity). Others are used for flow control (source quench) or routing (redirect).

An ICMPv4 message consists of an IPv4 packet header, followed by 8 bytes that specify the details for each ICMP message, followed by 32 or more bytes of data (depending on implementation).

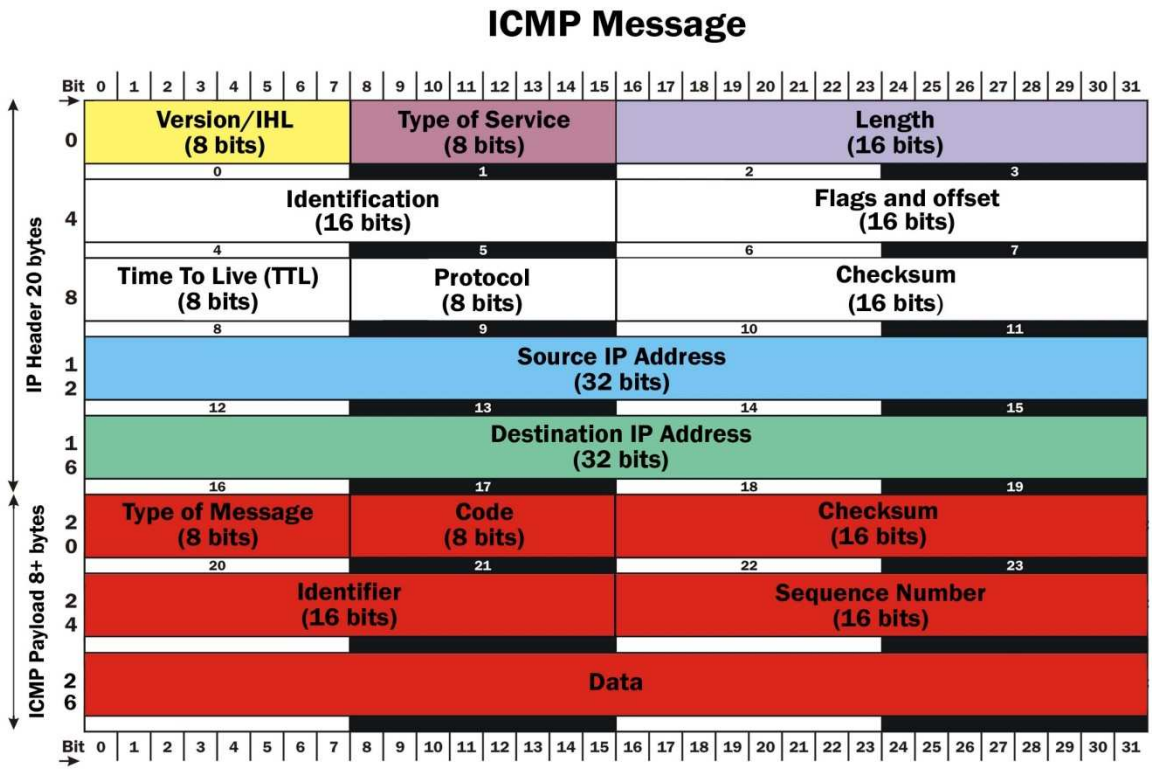


Figure 3.3-c: ICMPv4 Message Synax

The IP Header *Version* field contains the value 4 (for IPv4), and the *Header Length* is 20 bytes.

The IP Header *Type of Service* contains the value 0.

The IP Header *Total Length* field contains the sum of 20 (header length) + 8 (ICMP header Length) + number of bytes of data to be sent in message.

The IP Header *Time To Live* field is set to some reasonable count (or very specific counts if used to implement the *traceroute* function).

The IP Header *Protocol* field contains the value 1 (ICMPv4 protocol).

The IP Header *Source Address* field contains the IPv4 address of the sending node.

The IP Header *Destination Address* field contains the IPv4 address of the receiving node.

The ICMP Header *Type of Message* field (8 bits) specifies the ICMP message type, such as 8 for *Echo Request*. See below for the most common message types.

The ICMP Header *Code* field (8 bits) specifies options for the ICMP message. For example, with Message Type 3, the code defines what failed, e.g. 0 means “Destination network unreachable”, while 1 means “Destination host unreachable”.

The ICMP Header *Checksum* (16 bits) field is defined the same way as for an IPv4 header, but covers the bytes in the ICMP message (not including the IP header bytes).

The ICMP Header *ID* field (16 bits) can contain an ID, used only in Echo messages.

The ICMP Header *Sequence* field (16 bits) contains a sequence number, also used only in Echo messages.

Type	**Code**	**Description**
0 – Echo Reply	0	Echo reply (in response to Echo request)
3 – Destination Unreachable	0	Net unreachable
	1	Host unreachable
	2	Protocol unreachable
	3	Port unreachable
	4	Fragmentation required, and DF flag set
	5	Source route failed
4 – Source Quench	0	Source quench for congestion control
5 – Redirect Message	0	Redirect Datagram for the Network
	1	Redirect Datagram for the Host
	2	Redirect Datagram for the TOS & network
	3	Redirect Datagram for the TOS & host
8 – Echo Request	0	Echo Request

11 – Time Exceeded	0	TTL expired in transit
	1	Fragment reassembly time exceeded

For a ping diagnostic, the sending node transmits an ICMP Echo Request message (Type = 8). The ID can be set to any value (0 to 65,535), and the sequence number is set initially to 0, then is incremented by one for each ping in a sequence. The Data field (following the ICMP header) can contain any data (typically some ASCII string). When the receiving node gets an ICMP Echo Request, it sends an ICMP Echo Reply (Type = 0). The ID, sequence number and data fields in the reply must contain exactly what were sent in the request.

If the destination of a packet is unreachable, your TCP/IP stack will return a Destination Unreachable ICMP packet, with the code explaining what could not be reached.

If a packet cannot be sent by the preferred path (e.g. due to a link specified in a static route being down), an ICMP Redirect message will be sent to the packet sender (typically the previous router) which should then try other paths.

If the TTL in a packet header is decremented all the way to zero, the packet is discarded, and a Time Exceeded ICMP message will be sent to the packet sender.

If a node is receiving packets faster than it can handle them, it can send an ICMP Source Quench message to the sender, who should slow down.

According to the standards, all nodes should always respond to an Echo Request with an Echo Reply. Due to use of this function by many hackers and worms (for network mapping), many sites now violate the standard and do not reply to Echo requests. Many ISPs now actually block Echo Requests. Note that in IPv6, you cannot just block all ICMPv6 messages, as it is a far more integral part of the protocol.

3.3.5 – IPv4 Routing

TCP/IP was designed from the beginning to be an *internetworking* protocol. That means it supports ways to get packets from one node to another, even across multiple networks, by various routes through a possibly complex series of interconnections. If one or more links go down, the packets may travel by another route. Even within a given group of packets (say, ones that constitute a long E-mail message), some of the packets may go by one route, and others by another. The process of determining a viable route (or routes) to get traffic from A to B is called *routing*. This is one of the most complex areas of TCP/IP. There are entire long books on the subject. We will be covering only the simplest details, in order to show what how routing differs between IPv4 and IPv6.

Some simpler network protocols (such as Microsoft's NetBIOS or NetBEUI) are *non-routing*. They will work only within a single LAN. TCP/IP and Netware's IPX/SPX support *routing*. You can connect multiple networks together with them and any node in any network can (in general) exchange data with any other node in any connected network. The Internet is simply the largest set of interconnected networks in the world. TCP/IP's flexible routing capabilities are one of the things that make it possible.

There are many components used to create IP based networks, including NICs, cables, bridges, switches, and gateways. Of these, only gateways (network devices that can *forward* packets from one network segment to another) do routing. There are several kinds of gateways. The simplest case is a *router*, which uses various protocols, such as RIP, OSPF and BGP to determine where to forward packets, depending on their destination address. It is possible to build a router from a generic PC (or other computer) if it has multiple network interfaces (NICs), connected to multiple (otherwise disjoint) networks. Most operating systems with network support can be configured to do packet forwarding (accepting a packet from one network, via one NIC, and then forwarding it on to another network, via a different NIC). Typically no changes are made to the packet other than decrementing the *hop count* in the IP packet header. If NAT is being performed, numerous changes may be made to the packet header.

It is also possible for a gateway node to do other processing as the packets flow through it, such as filtering packets on certain criteria (e.g. allow traffic using port 25 to node 172.20.0.11 to pass, but block port 25 traffic to all other nodes). These are called *packet filtering firewalls*. They are really just routers that allow more control over the flow of traffic, and can protect the network from various attacks. Even in a packet filtering firewall, all processing still takes place at the *Internet Layer*. More sophisticated packet filtering firewalls can "inspect" the contents of the packets and maintain a record ("state") of things that really are associated with higher levels of the network stack (e.g. *Transport* or *Application* Layers). This is called *deep packet inspection*, or *stateful inspection*.

It is also possible to have a bastion host that doesn't just forward traffic; it receives protocol connections on behalf of nodes on the internet network, and relays them onward if they are acceptable. They act as a *proxy* for the internal servers. Processing here takes place at the *Application Layer*. Proxy firewalls are much more secure, but also more complex and slower. Typically, a proxy server must be created for *each protocol* handled by the firewall. There can be both incoming proxies (as described above) and outgoing proxies (your node makes an outgoing connection to a proxy in your firewall, and it makes a further outgoing connection to the node you really want to connect to. These allow better control than a simple packet filtering firewall. If a firewall does both packet forwarding with stateful inspection, *and* has proxy servers (incoming and/or outgoing) for at least some protocols, it is called a *hybrid* firewall, and can provide the best of both worlds.

The standards relevant to routing in IPv4 include:

- **RFC 1058, "Routing Information Protocol", June 1988 (Historic)**
- RFC 1142, "OSI IS-IS Intra-domain Routing Protocol", February 1990 (Informational)
- RFC 1195 "Use of OSI IS-IS for Routing in TCP/IP and Dual Environments", December 1990 (Standards Track)
- **RFC 2328, "OSPF Version 2", April 1998 (Standards Track)**
- **RFC 2453, "RIP Version 2", November 1998 (Standards Track)**
- **RFC 4271, "A Border Gateway Protocol 4 (BGP-4)", January 2006 (Standards Track)**

In Windows, you can view all currently known routes with the "route print" command:

```
C:>route print
===========================================================================
Interface List
```

```
 11...00 22 15 24 32 9c ......Realtek PCIe GBE Family Controller
  2...00 22 b0 51 37 7c ......D-Link DGE-530T Gigabit Ethernet Adapter
  1...........................Software Loopback Interface 1
 12...00 00 00 00 00 00 00 e0 Microsoft ISATAP Adapter
 14...00 00 00 00 00 00 00 e0 Microsoft ISATAP Adapter #2
 19...00 00 00 00 00 00 00 e0 Teredo Tunneling Pseudo-Interface
===========================================================================

IPv4 Route Table
===========================================================================
Active Routes:
Network Destination        Netmask          Gateway       Interface  Metric
          0.0.0.0          0.0.0.0      172.20.0.1     192.168.1.8    276
        127.0.0.0        255.0.0.0         On-link         127.0.0.1    306
        127.0.0.1  255.255.255.255         On-link         127.0.0.1    306
  127.255.255.255  255.255.255.255         On-link         127.0.0.1    306
      192.168.0.0      255.255.0.0         On-link       192.168.1.8    276
      192.168.2.1  255.255.255.255         On-link       192.168.1.8    276
  192.168.255.255  255.255.255.255         On-link       192.168.1.8    276
        224.0.0.0        240.0.0.0         On-link         127.0.0.1    306
        224.0.0.0        240.0.0.0         On-link       192.168.1.8    276
  255.255.255.255  255.255.255.255         On-link         127.0.0.1    306
  255.255.255.255  255.255.255.255         On-link       192.168.1.8    276
===========================================================================
Persistent Routes:
  Network Address          Netmask  Gateway Address  Metric
          0.0.0.0          0.0.0.0      192.168.0.1  Default
===========================================================================
```

There are a number of routing protocols for IPv4 that are typically handled only in the core, or where a customer network meets the core, the *edge router*. These include RIP, RIPv2, EIGRP, IS-IS, OSPF and BGP.

Routing is a very deep, complex subject, and we will be touching only on the most obvious aspects in this book, to give a rough idea of the differences in routing between IPv4 and IPv6.

RIP –Routing Information Protocol, version 1. Defined in RFC 1058, “Routing Information Protocol”, June 1988. This protocol is an older one, but still in widespread use (especially by low end routers, such as SOHO units and in some interior gateways). It is used to exchange routing information with gateways and other hosts. It is based on the *distance vector* algorithm, which was first used in the ARPANET, circa 1967. RIP is a UDP based protocol, using port 520.

RIPv2 – Routing Information Protocol, version 2. Defined in RFC 2453, “RIP Version 2”, November 1998. Although OSPF and IS-IS are superior, there were so many implementations of RIP in use, it was decided to try to improve on it. Extensions were made to incorporate the concepts of autonomous systems, IGP/EGP interactions, subnetting and authentication. The lack of subnet masks in RIPv1 was a particular problem. RIPv2 is limited to networks whose longest routing path is 15 hops. It also uses fixed “metrics” to compare alternative routes, which is an oversimplification. However, RIPv2 becomes unstable if you try to account for different metrics. See RFC for details.

EIGRP – Enhanced Interior Gateway Routing Protocol. This is *not* an IETF protocol, but a Cisco proprietary routing protocol based on their earlier IGRP. EIGRP is able to deal with CIDR, including use of variable length subnet masks. It can run separate routing processes for IPv4, IPv6, IPX and AppleTalk protocols, but does not support translation between protocols. For details, see Cisco documentation.

IS-IS – Intermediate System to Intermediate System Routing Protocol. IS-IS (pronounced "eye-sys") was originally developed by Digital Equipment Corporation (DEC) as part of DECnet Phase V, and formally defined as part of ISO/IEC 10589:2002 for the Open System Interconnection reference design. It is *not* an Internet standard, although the details are published as *Informational* RFC 1142, "OSI IS-IS Intra-domain Routing Protocol", February 1990. Another RFC specifies how to use IS-IS for routing in TCP/IP and/or OSI environments: RFC 1195 "Use of OSI IS-IS for Routing in TCP/IP and Dual Environments", December 1990. IS-IS is an Interior Gateway Protocol, for use within an administrative domain or network. It is not intended for routing between autonomous systems, which is the role of BGP. It is not a distance vector algorithm, it is a *link-state* protocol. It operates by reliably flooding network topology information through a network of routers, allowing each router to build its own picture of the complete network. OSPF (developed by the IETF about the same time) is more widely used, although it appears that IS-IS has certain characteristics that make it superior in large ISPs.

OSPFv2 – Open Shortest Path First, Version 2. Unlike EIGRP and IS-IS, OSPF is an IETF standard. OSPFv2 is defined in RFC 2328, "OSPF Version 2", April 1998. OSPF is the most widely used Interior Gateway Protocol today (as opposed to BGP, which is an Exterior Gateway Protocol). Like IS-IS, OSPF is a link-state protocol. It gathers link state information from available routers and builds a topology map of the network. It was designed to support variable-length subnet masking (VLSM) or CIDR addressing models. Changes to the network topology are rapidly detected, and it converges on a new optimal routing structure within seconds. It allows specification of different metrics ("cost of transmission" in some sense) for various links to allow better modeling of the real world (where some links are fast, some slow). OSPF does not layer over UDP or TCP, but uses IP datagrams with a protocol number of 89. This is very different from RIP or BGP. OSPF uses multicast, including the special addresses:

224.0.0.5	All SPF/link state routers	AllSPFRouters
224.0.0.6	All Designated Routers	ALLDRouters

For routing IPv4 multicast traffic, there is MOSPF (Multicast Open Short Path First), defined in RFC 1584, "Multicast Extensions to OSPF", March 1994. However, this is not widely used. Instead, most people use PIM in conjunction with OSPF or other Interior Gateway Protocols.

BGP-4 – Border Gateway Protocol 4. Defined in RFC 4271, "A Border Gateway Protocol 4 (BGP-4)", January 2006. This version supports routing only IPv4. There are defined multiprotocol extensions (BGP4+) that support IPv6 and other protocols, which will be described in Chapter 5.

BGP is an External Gateway Protocol (compare with IS-IS and OSPFv2, which are Internal Gateway Protocols). It is not used within networks, but only between Autonomous Systems. Its primary function is to exchange network reachability information with other BGP systems. This includes information on the list of Autonomous Systems (ASes) that reachability information traverses. This is sufficient for BGP to construct a graph of AS connectivity from which routing loops can be pruned, and, at the AS level certain policy decisions may be enforced.

BGP-4 includes mechanisms for supporting CIDR. They can advertise a set of destinations as an IP prefix, eliminating the concept of network "class" which was present in early BGP implementations. BGP-4 also has mechanisms that allow aggregation of routes and AS paths. Most networks that obtain service from ISPs never deploy BGP themselves. It is mostly for exchange of information *between* ISPs, especially if they are multi-homed (obtain upstream service from more than one source). This would be referred to as *Exterior Border Gateway Protocol* or EBGP. Enormous networks that are too large for OSPF could deploy BGP themselves as a top level linking multiple OSPF routing domains (this would normally be referred to as *Interior Border Gateway Protocol* or IBGP).

BGP is a *path vector* protocol. It does not use IGP metrics, but makes routing decisions based on path, network policies and/or rulesets. It replaces the now defunct Exterior Gateway Protocol (EGP), which was formally specified in RFC 904, "Exterior Gateway Protocol Formal Specification", April 1984.

3.3.5.1 – Network Address Translation (NAT)

It is also possible for a gateway to do Network Address Translation (NAT) as packets are forwarded. One form of this allows multiple internal nodes (which use Private Addresses, such as 10.1.2.3) to be translated to *globally routable* addresses (like 123.45.67.89) on the way out. It also can translate the globally routable destination address of packets sent in reply to an outgoing packet back to the Private Address of the originating node, so that it can complete a query/response transaction. Typically ports are shifted by a NAT gateway in such a way that it can figure out which internal node to send reply packets to. This allows many internal nodes to "share" (hide behind) a single globally routable IPv4 address (necessary now that we are running out of these). NAT will be covered in more detail in the next chapter.

The relevant RFCs for NAT for IPv4 include:

- **RFC 1918, "Address Allocation for Private Internets", February 1996 (Best Current Practices)**
- **RFC 2663, "IP Network Address Translator (NAT) Terminology and Considerations", August 1999 (Informational)**
- RFC 2694, "DNS extensions to Network Address Translators (DNS_ALG)", September 1999 (Informational)
- RFC 2709, "Security Model with Tunnel-mode IPsec for NAT Domains", October 1999 (Informational)
- **RFC 2993, "Architectural Implications of NAT", November 2000 (Informational)**
- **RFC 3022, "Traditional IP Network Address Translator (Traditional NAT)", January 2001 (Informational)**
- RFC 3235, "Network Address Translator (NAT)-Friendly Application Design Guidelines", January 2002 (Informational)
- RFC 3519, "Mobile IP Traversal of Network Address Translation (NAT) Devices", April 2003 (Standards Track)
- **RFC 3715, "IPsec-Network Address Translation (NAT) Compatibility Requirements", March 2004 (Informational)**
- **RFC 3947, "Negotiation of NAT-Traversal in the IKE", January 2005 (Standards Track)**

- RFC 4008, “Definitions of Managed Objects for Network Address Translators (NAT)”, March 2005 (Standards Track)
- **RFC 4787, “Network Address Translation (NAT) Behavioral Requirements for Unicast UDP”, January 2007 (Best Current Practices)**
- RFC 4966, “Reasons to Move the Network Address Translator – Protocol Translator (NAT-PT) to Historic Status” (Informational)
- RFC 5128, “State of Peer-to-Peer (P2P) Communication across Network Address Translators (NATs)”, March 2008 (Informational)
- RFC 5207, “NAT and Firewall Traversal Issues of Host Identity Protocol (HIP) Communication”, April 2008 (Informational)
- **RFC 5382, “NAT Behavioral Requirements for TCP”, October 2008 (Best Current Practices)**
- **RFC 5389, “Session Traversal Utilities for NAT (STUN)”, October 2008 (Standard Track)**
- **RFC 5508, “NAT Behavioral Requirements for ICMP”, April 2009 (Best Current Practices)**
- RFC 5597, “Network Address Translation (NAT) Behavioral Requirements for the Datagram Congestion Control Protocol”, September 2009 (Best Current Practices)
- **RFC 5684, “Unintended Consequences of NAT Deployments with Overlapping Address Space”, February 2010 (Informational)**

It should be obvious from the number of RFCs that explain how NAT affects other things that NAT has a heavy impact on almost every aspect of networks. There are also a lot of “Informational” RFCs required to explain exactly *how* it impacts these things. Removing NAT has no downside (given sufficient addresses) and vastly simplifies network architecture and management in addition to lowering costs. It also vastly simplifies application design and implementation. The removal of NAT and restoration of the flat address space is one of the main benefits of moving to IPv6. Unfortunately, we have an entire generation of network engineers who have assumed that NAT is “the way networks are done”, and don’t realize it was created only as a temporary crutch to extend the life of the IPv4 address space until IPv6 could be completed and deployed. Before NAT, the IPv4 Internet was “flat” and firewalls had very effective security without NAT (I call this “classic firewall architecture”). In IPv6, we are simply returning to the original concept of any node to any node connectivity that characterized the Pre-NAT IPv4 Internet. Protocols like SIP, IPsec, IKE and Mobile IP will work far better without NAT in the way. DNS is also greatly simplified in the absence of NAT (no internal vs. external “views” are required).

Most routers and firewalls typically include NAT for IPv4, although it would be possible to have a NAT gateway without any filtering or routing capabilities that does *only* NAT.

In general, any gateway that modifies the source and/or destination addresses in a packet (possibly also the source port number) is doing NAT. There are several forms of it, the most popular being *address masquerading* (*hide mode* NAT) and *one-to-one* (BINAT, or *static* NAT).

Most IPv4 networks today make use of *private addresses* as defined in RFC 1918, “Address Allocation for Private Internets”, February 1996. Basically, three blocks of addresses (10.0.0.0/8, 172.16.0.0/12 and 192.168.0.0/16) were permanently removed from the available Internet allocation pool, marked as “unroutable” on the Internet, and reserved for use as something similar to telephone extension numbers in an office (hiding behind a single company phone number, via a Private Branch Exchange). It is possible for any company to use addresses from any or all of these ranges to number the nodes inside their networks. However, these addresses cannot be routed on the Internet from anyone, since they are

no longer globally unique). Hence, if the users of nodes with those addresses want to use the Internet, there must be address translation to and from "real" (globally unique) addresses at the gateway that connects them to the Internet, which is what NAT does.

Note that the most popular form of NAT is more properly called NAPT ("Network Address Port Translation") which involves the translation of both IP addresses and port numbers.

NAT is defined in RFC 3022, "Traditional IP Network Address Translator (Traditional NAT)", January 2001. Some aspects of NAT are defined in RFC 2663, "IP Network Address Translator (NAT) Terminology and Considerations", August 1999.

One form of NAT Traversal (STUN) is defined in RFC 5389 "Session Traversal Utilities for NAT (STUN)", October 2008. STUN is a protocol that serves as a tool for other protocols in dealing with Network Address Translator (NAT) traversal. It can be used by an endpoint to determine the IP address and port allocated to it by a NAT. It can also be used to check connectivity between two endpoints, and as a *keepalive* protocol to maintain NAT bindings. STUN works with many existing NATs and does not require any special behavior from them.

Connection without NAT (inside the LAN)

Say you have two nodes (Alice and Bob) on your LAN. Alice has the address 10.50.3.12, and Bob has the address 10.50.3.75 (both private addresses). They can make connections within their LAN (to any address in the 10.0.0.0/8 network) with no problem. Say there is a web server (port 80) at 10.1.20.30. In the following, we will specify the port number appended to the IP address, separated by a colon (e.g. 10.50.3.12:12345). When Alice makes a connection to the web server, the *destination* port is 80, but her *source* port is a randomly chosen value greater than 1024, that is not already in use (e.g. 12345 or 54321). The same source port would be used for the duration of the connection. Replies from the server would be sent using Alice's source address and port as the *destination* address and port in the reply packets. For example:

	Source Addr:Port	Destination Addr:Port
Traffic from Alice to Server:	10.50.3.12:12345	10.1.20.30:80
Replies from Server to Alice:	10.1.20.30:80	10.50.3.12:12345
Traffic from Bob to Server:	10.50.3.75:54321	10.1.20.30:80
Replies from Server to Bob:	10.1.20.30:80	10.50.3.75:54321

Note: the above behavior is somewhat simplified. Such as server could accept only one connection at a time, which would have to complete before anyone else could connect. This is because a given address:port can only handle one connection at any given time. A real-world server would have a parent process listening for connections on a well known port (e.g. 80). When some client connects to the well known port, the parent process would create a child process (or thread) which would accept the connection (using yet another unused port number) and process it. Meanwhile the main process would go back to listening for further connections on the well known port. If ten users were connected at a time, there would be 11 processes running, one main process and ten child processes (one for each connection). From the viewpoint of the client (e.g. with "netstat –na") it would appear that the remote port (the one on the server) was the original well known port (e.g. 80).

Connection through Hide Mode NAT

But how do Alice and Bob connect to *www.ipv6.org*? That node happens to have an IPv4 address of 130.237.234.40, and we're still in Chapter 3 (about IPv4), so they don't have IPv6 yet! Let's say there is a NAT gateway where their LAN (or ISP) connects to the Internet. It has an "outside" address (which must be a valid, routable IPv4 address) of 12.34.56.137. If *either* Alice or Bob connect to *www.ipv6.org* (over IPv4), the webpage there will indicate to both of them that they are connecting over IPv4, from the address 12.34.56.137, not from their respective private addresses, even if the connections are made at the same instant. How can *www.ipv6.org* reply with the correct webpage to each of them?

With hide mode NAT, the gateway is translating the source address in Alice's packets from 10.50.3.12 to 12.34.56.137. It is *also* translating the source address in Bob's packets from 10.50.3.75 to 12.34.56.137. The destination address was 130.237.234.40:80 for both Alice and Bob. Their browsers would each choose a random source port. Let's say Alice's chose 10123 and Bob's chose 20321. The NAT gateway would not only translate the source address from both Alice and Bob, it would also shift the source ports, and keep track of that shift in a table which contains the source address, the original source port, and the shifted source port (for each connection). Let's say Alice's port is shifted to 30567 and Bob's to 40765. The New Source Address for outgoing connections and the old destination address for incoming connections will always be the same (the outside address of the NAT gateway), so it does not need to keep those in the table. The resulting NAT table would look like this:

Source Addr:Port	Shifted Port
10.50.3.12:10123	30567
10.50.3.75:20321	40765

	Src Addr:Port	Xlat Src Addr :Port	Dest. Addr :Port
Traffic from Alice to server:	10.50.3.12:10123	12.34.56.137:30567	130.237.234.40:80
	Src Addr:Port	Dest Addr:Port	Xlat Dest Addr:Port
Replies from server to Alice:	130.237.234.40:80	12.34.56.137:30567	10.50.3.12:10123
	Src Addr:Port	Xlt Src Addr :Port	Dest. Addr :Port
Traffic from Bob to server:	10.50.3.75:20321	12.34.56.137:40765	130.237.234.40:80
	Src Addr:Port	Dest Addr:Port	Xlt Dest Addr:Port
Replies from server to Bob:	130.237.234.40:80	12.34.56.137:40765	10.50.3.75:20321

Alice's connection to *www.ipv6.org* appears to be coming from 12.34.56.137:30567. When www.ipv6.org replies to Alice, it is sent from 130.237.234.40:80 to 12.34.56.137:30567. The NAT gateway looks up that port in its table, and sees that it was used for an outgoing connection from 10.50.3.12:10123, so it translates the destination address and port to Alice's private address and port, thereby forwarding the packets correctly to Alice.

Bob's connection to *www.ipv6.org* appears to be coming from 12.34.56.137:40765. When www.ipv6.org replies to Bob, it is sent from 130.237.234.40:80 to 12.34.56.137:40765. The NAT gateway looks that port up in its table, and sees that it was used from a connection from 10.50.3.75:20321, so it translates

the destination address and port to Bob's private address and port, thereby forwarding those packets correctly to Bob.

BINAT (One to One NAT)

If you are doing NAT at your gateway, most routers or firewalls support another form of NAT, which is known as BINAT (Bidirectional NAT) or One-to-One (sometimes also *static* NAT). This works much the same as regular (hide mode) NAT, except there is no port shifting involved. This means there can only be *one* internal node associated with each globally routable external address. This is used only for servers that must be accessible from the outside world.

Typically a server has both an *internal* (private) address (e.g. 10.0.0.13), and an *external* (unique, globally routable) address (e.g. 12.34.56.131). With outgoing connections, the gateway rewrites the source address of each packet to be the external address for that node (but does not shift the port). For incoming connections, the gateway rewrites the destination address to be the internal address for that node. Internally, the node will have only the internal address. However, if you connected to *www.ipv6.org* from such a node, the resulting webpage would show a connection not from the *internal* address of the server, but from the unique external address associated with that node. This is similar to hide-mode NAT except that there is exactly one internal node per external address (rather than many), there is no port shifting, and the mapping can be done in both directions (incoming and outgoing connections).

There is a minor problem of the *missing arp* that must be solved in some way for this to work (there is no physical node at the external address, so no node will respond to ARP requests concerning that address). One approach is to configure a static arp on the gateway that can supply that response. Every operating system or router has some way to do this. Without that, connections from the outside will not work. It is also possible in most cases to assign the external address as an *alias* to the outside interface of the gateway (in addition to its real address). Solving the *missing arp* problem is one of the most difficult things for firewall administrators to master. This problem only exists in IPv4, as no NAT is needed or done in IPv6, hence no missing arp (actually in IPv6, it would be a missing ND response).

BINAT at least allows incoming connections, but uses up one globally routable IPv4 address for each server node. Most SOHO gateways do not support BINAT. Many do have a simpler mechanism called *port redirection*, which allows incoming connections to the hide mode external address. At most one internal server can be configured as the target for any given port. So, you could configure an internal mail server and redirect ports 25 (SMTP), 110 (POP3) and 143 (IMAP) to it. However, if you have two internal web servers both configured for port 80, you could not redirect port 80 on the gateway to *both* servers.

Ramifications of Using NAT

When network address translation happens, the NAT gateway is actually rewriting new values into the address and port number fields in the IP and TCP (or UDP) packet headers of all packets flowing through the NAT gateway, according to the rule just specified. For outgoing packets, it is rewriting the source address and source port. For incoming packets, it is rewriting the destination address and destination port. Obviously, this would invalidate the IP and TCP header checksums (the IP header contains source and destination addresses, the TCP header contains the source and destination port numbers).

Therefore the NAT gateway *also* has to recalculate both IP and TCP header checksums and rewrite *those* as well.

Packet fragmentation is a real complication for TCP and UDP via NAT. A NAT gateway must reassemble an entire packet, in order to be able to recalculate the TCP checksum (which covers all bytes in the payload, plus the pseudo header which contains the source and destination addresses). It typically must then re-fragment the packet for further transmission.

What about IPsec Authentication Header? (Note: IPsec will be discussed in detail in Chapter 6.) The IPsec AH algorithm works like a checksum, but there is a key which only the sender has, required to generate the cryptographic checksum. All this address and port rewriting invalidates the existing AH cryptographic checksum, and the NAT gateway does **not** have the necessary key to regenerate a correct new AH for the modified packet headers. Because of this IPsec does not work through a NAT gateway. Actually AH is performing its function very effectively; it **is** detecting tampering with the contents of the packet header! It just happens that this tampering is done by a NAT gateway, not a hacker. It's kind of like getting hit by "friendly fire" in a war zone (getting shot by your own side). If any node other than the original sender could generate a new valid AH checksum, then AH would not be very useful! IPsec and NAT are mutually exclusive (although IPsec VPNs can be made to work in conjunction with NAT Traversal).

Another ramification involves FTP (File Transfer Protocol). FTP is a very old protocol. In active mode, FTP uses separate connections for control traffic (commands) and for data traffic. The initiating host identifies the corresponding data connection with its network layer and transport layer addresses. Unfortunately NAT invalidates this. Fortunately here, it is possible to create a *reverse FTP proxy* (included on most firewalls) that solves this problem. Without such a proxy though, FTP will not work if NAT is in place, even for outgoing connections. My company early on ported a popular one for IPv4 to IPv6. That allowed FTP connections to dual stack networks such as freebsd.org to work from our own dual stack network.

Peer-to-peer applications have the same kinds of problems with NAT. You must somehow provide a way for your peers to connect to *you* for these applications to work. All participants really need a real, globally routable IP address. This is not easy to arrange on the First Internet.

The SIP (Session Initiation Protocol) is used with many things, including VoIP and video conferencing. It also has major problems with NAT. SIP may use multiple ports to set up a connection and transmit the analog stream over RTP (Real Time Protocol). IP addresses and port numbers are encoded in the payload and must be known prior to the traversal of NAT gateways. Again, a SIP proxy on the gateway can help resolve this problem. Another solution is to use NAT traversal, such as STUN. Unfortunately, in these days of widespread NAT, both the caller and the callee are typically behind NAT, so VoIP must overcome problems with NAT both *going out* from the caller, and *coming in* to the callee.

Another problem with NAT is the limit of 65,536 ports on the NAT gateway. When NAT was first deployed, most network applications used only one or two ports. Some recent applications (Apple's iTunes and Google Maps) use 200-300 ports at a time. If each node is using 300 ports, then there can be at most 200 nodes behind a given external IPv4 address. If the NAT gateway runs out of ports, there can be very mysterious failures in network applications. For example, in Google Maps, some areas of the map never get drawn. There is no way for end users (or typically even the network administrator) to

determine that this has happened other than by seeing mysterious failures in some applications. This means that a larger number of NAT gateways (and valid external IPv4 addresses) are required today than in the past, for a given number of users behind NAT. Just as we are running out of them!

Most legacy applications (like web surfing and E-mail) work OK through one layer of NAT. Even with chat, today there must be an intermediary system that two or more chatters connect to via outgoing connections from their nodes (e.g. AOL Instant Messenger). In a flat address space (especially with working multicast), much better connectivity models are possible, that may require little or no central facilities.

As the IPv4 addresses run out, it will become more common to have *multiple layers* of NAT. This can happen today, if you deploy a Wi-Fi access point with NAT behind a DSL modem that *also* has NAT. If you think a single layer of NAT causes problems, you should try dealing with multiple layers of it!

With the wide scale deployment of NAT, we have lost the original end-to-end model of the early First Internet, which was a core feature. Today users are either *content producers* who can publish information or videos (e.g. cnn.com, youtube.com) or *content consumers* who can view the content published by the producers. It is much more complicated and expensive to be a producer in the current First Internet (with NAT), than to be a consumer. There are relatively few producers, and millions of consumers. This was not that much of a problem when most people were running mainly web browsers and E-mail clients on their nodes. As newer applications emerge (VoIP, IPTV, multi-player games, Peer-to-peer), this new "digital divide" between producers and consumers is becoming more of a problem. Today, many people would like to be *prosumers* (both producers and consumers of content).

All of these problems go away with a flat address space (no NAT). Unfortunately, there is no way to restore the flat address space of the early (pre-NAT) First Internet. The First Internet is now permanently broken (there are not enough addresses to allow even the existing users to have access without NAT, even if we used all of the remaining unallocated addresses today). The only real solution is to switch to IPv6 (at least for protocols such as VoIP, P2P, multiplayer games, IPTV and IPsec VPNs).

3.3.5.2 – Basic IPv4 Routing

In the simplest case, where two nodes (A and B) are on the same network segment (not separated by any router), no routing is required. Let's say node A want to send a packet to node B. Node A determines if node B is in the same network segment by examining B's IP address and the network subnet mask. If node B is in the same subnet as A's IP address, then B is a *local node*. Node A simply uses B's MAC address from its ARP table to send the packet to B. If there is no entry for B's IP address, then node A does *address resolution* (obtains the MAC address for B), as described earlier.

If B's address is *not* in the local subnet, B is *not* a local node, and the packet (with B's correct IP address as the destination) is sent to the node which serves as the *default gateway* for A's subnet (A may first have to do an ARP to obtain the MAC address of the default gateway). The default gateway is a node with multiple network interfaces that knows how to forward the packet on towards the network in which B's IP address is found. Note that by default, *packet forwarding* (relaying packets from one interface to another on a multi-homed system) is not enabled. It must be specifically enabled for each protocol (IPv4 and IPv6). The address of a network's default gateway is known to every node in a subnet,

either through manual configuration or via DHCPv4. Once the default gateway receives the packet, it may already have the necessary routing information to know where to send that packet (either via *static routes*, or via a routing protocol, such as RIP, OSPF and/or BGP). In the case of a home network, your SOHO router typically just knows how to forward packets for the outside world to yet another gateway at the ISP, where the real routing takes place (via its *own* default gateway, which is a node at the ISP).

Once your traffic gets to your default gateway, that node typically uses an Interior Gateway Routing Protocol (RIP, RIPv2 or OSPF) to route that traffic to the edge of your overall network (e.g. the place your organization's or ISP's network connects to the rest of the Internet). At that point, an Exterior Gateway Routing Protocol (typically BGP-4) is used to determine the best route to the correct edge router for the destination address. Once your traffic arrives there, once again an Interior Gateway Routing Protocol (RIP, RIPv2 or OSPF) takes over and gets the packets to the default gateway of the subnet where the destination node lives. From there, ARP is used to forward the packets to the actual destination node, because the default gateway and the destination node are now on the same subnet. And all this takes place in the blink of an eye, billions of times a day, just like clockwork.

3.4 – TCP: The Transmission Control Protocol

The Transmission Control Protocol is defined in RFC 793, "Transmission Control Protocol", September 1981. This is a *Transport Layer* protocol. TCP implements a *reliable*, *connection oriented* model. When we say *reliable*, we aren't talking about a well designed or robust protocol. With respect to TCP, "reliable" simply means that it includes error detection and recovery (via retransmission). The term *connection oriented* refers to the fact that TCP is designed to handle potentially large streams of data (typically larger than a single packet). It does this by breaking the large object up into multiple packet sized chunks and sending them out in an ordered sequence. For example, a large E-mail message or a JPEG photograph might require quite a few packets. Software that uses TCP typically *opens* (initiates) a connection for I/O, reads and/or writes potentially a lot of data to it, then when done, *closes* (terminates) the connection. This is very similar to the process for reading and writing files, and in fact in UNIX, network streams are just a special kind of file.

The following standards are relevant to TCP:

- **RFC 793, "Transmission Control Protocol", September 1981 (Standards Track)**
- RFC 896, "Congestion Control in IP/TCP Internetworks", January 1984 (Unknown)
- **RFC 1001, "Protocol standard for a NetBIOS service on a TCP/UDP transport: Concepts and Methods", March 1987 (Standard Track)**
- **RFC 1002, "Protocol standard for a NetBIOS service on a TCP/UDP transport: Detailed specifications", March 1987 (Standards Track)**
- RFC 1006, "ISO Transport Service on top of the TCP Version: 3", May 1987 (Standards Track)
- RFC 1085, "ISO presentation services on top of TCP/IP based internets", December 1998 (Uknown)
- RFC 1086, "ISO-TP0 bridge between TCP and X.25", December 1988 (Unknown)
- **RFC 1144, "Compressing TCP/IP Headers for Low-Speed Serial Links", February 1990 (Standards Track)**

- RFC 1155, “Structure and identification of management information for TCP/IP-based internets”, May 1990 (Standards Track)
- **RFC 1180, “TCP/IP tutorial”, January 1991 (Informational)**
- RFC 1213, “Management Information Base for Network Management of TCP/IP-based internets: MIB II”, March 1991 (Standards Track)
- RFC 1323, “TCP Extensions for High Performance”, May 1992 (Standards Track)
- RFC 2018, “TCP Selective Acknowledgement Options”, October 1996 (Standards Track)
- RFC 2126, “ISO Transport Service on top of TCP (ITOT)”, March 1997 (Standards Track)
- RFC 2883, “TCP Processing of the IPv4 Precedence Field”, June 2000 (Standards Track)
- RFC 2883, “An Extension to the Selective Acknowledgement (SACK) Option for TCP”, July 2000 (Standards Track)
- RFC 2988, “Computing TCP’s Retransmission Timer”, November 2000 (Standards Track)
- RFC 3042, “Enhancing TCP’s Loss Recovery Using Limited Transport”, January 2001 (Standards Track)
- RFC 3293, “General Switch Management Protocol (GSMP) Packet Encapsulation for Asynchronous Transfer Mode (ATM), Ethernet and Transmission Control Protocol (TCP)”, June 2002 (Standards Track)
- RFC 3390, “Increasing TCP’s Initial Window”, October 2002 (Standards Track)
- RFC 3517, “A Conservative Selective Acknowledgement (SACK)-based Loss Recovery Algorithm for TCP”, April 2003 (Standards Track)
- RFC 3782, “The New Reno Modifications to TCP’s Fast Recovery Algorithm”, April 2004 (Standards Track)
- RFC 3821, “Fiber Channel over TCP/IP (FCIP)”, July 2004 (Standards Track)
- RFC 4015, “The Eifel Response Algorithm for TCP”, February 2005 (Standards Track)
- **RFC 4022, “Management Information Base for the Transmission Control Protocol (TCP)”, March 2005 (Standards Track)**
- **RFC 4614, “A Roadmap for Transmission Control Protocol (TCP) Specification Documents”, September 2006 (Informational)**
- RFC 4727, “Experimental Values in IPv4, IPv6, ICMPv4, ICMPv6, UDP and TCP Headers”, November 2006 (Standards Track)
- RFC 4898, “TCP Extended Statistics MIB”, May 2007 (Standards Track)
- RFC 4996, “Robust Header Compression (ROHC): A Profile for TCP/IP (ROHC-TCP)”, July 2007 (Standards Track)
- RFC 5348, “TCP Friendly Rate Control (TFRC): Protocol Specification”, September 2008 (Standards Track)
- RFC 5482, “TCP User Timeout Option”, March 2009 (Standards Track)
- **RFC 5681, “TCP Congestion Control”, September 2009 (Standards Track)**
- RFC 5682, “Forward RTO-Recovery (F-RTO): An Algorithm for Detecting Spurious Retransmission Timeouts with TCP”, September 2009 (Standards Track)
- RFC 5734, “Extensible Provisioning Protocol (EPP) Transport over TCP”, August 2009 (Standards Track)

3.4.1 – TCP Packet Header

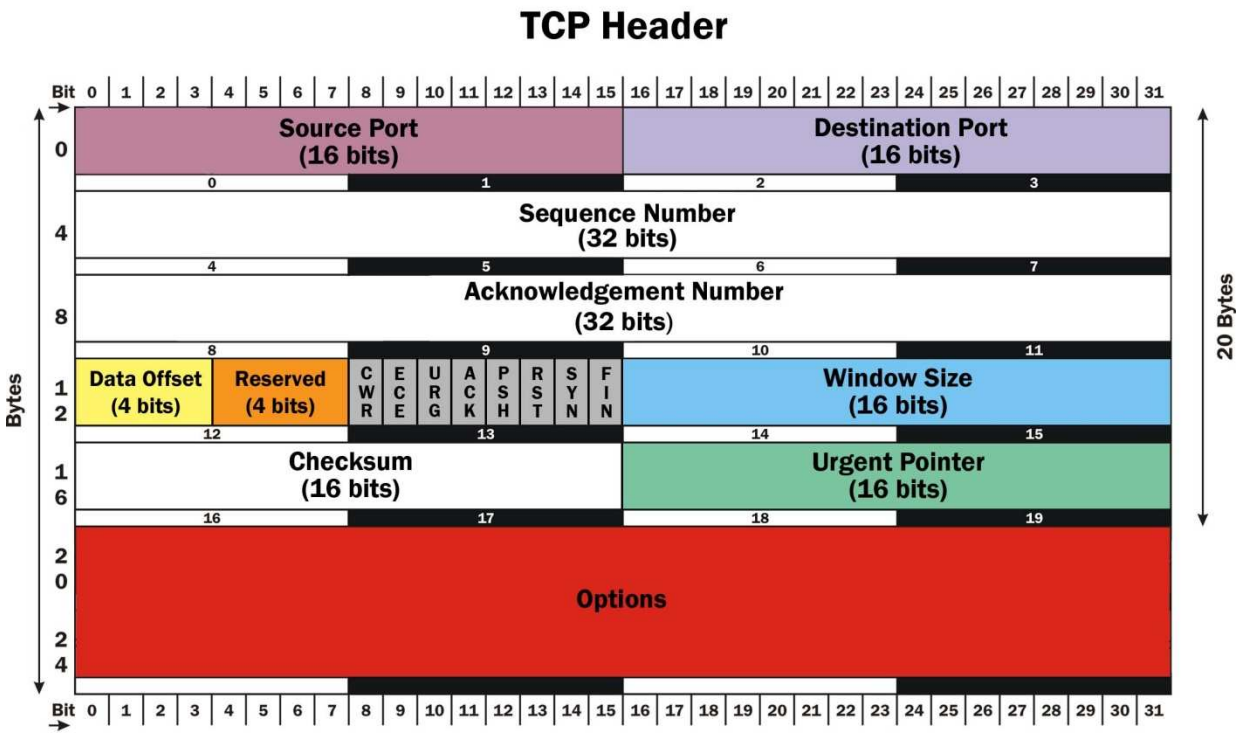


Figure 3.4-a: TCP Packet Header

Source Port (16 bits) – specifies the port that the data was written to on the sending node.

Destination Port (16 bits) – specifies the port that the data will be read from on the receiving node.

Sequence Number (32 bits) – meaning depends on the value of the SYN flag:

- If the SYN flag is set, this field contains the initial sequence number. The sequence number of the actual first data byte (and the acknowledgment number in the resulting ACK) will then be that value plus 1.
- If the SYN flag is clear, this field contains the accumulated sequence number of the first data byte of this packet for the current session.

Acknowledgement Number (32 bits) – used to acknowledge receipt of data:

- if the ACK flag is set, this field is the next sequence number that the receiver is expecting. This acknowledges receipt of all previous bytes.
- If the ACK flag is clear, this field is not used.

Data Offset (4 bits) – specifies the size of the TCP header in 32 bit words. The minimum value is 5 words (20 bytes), and the maximum value is 15 words (60 bytes), allowing for up to 40 bytes of options.

Reserved field (4 bits) – not currently used, and must be zeros.

There are eight 1-bit flags (8 bits total) as follows (in order from most significant bit to least significant bit):

- CWR – Congestion Window Reduced. If set by sender, indicates it has received a TCP segment with the ECE flag set, and has responded in congestion control mechanism.
- ECE – ECN Echo. If SYN flag is set, then ECE set indicates that the TCP peer is ECN capable. If SYN flag is clear, then ECE flag set indicates that a Congestion Experienced flag in the IP header set was received during normal transmission
- URG – indicates whether or not Urgent pointer field is significant
- ACK - if set, indicates that the Acknowledgement field is significant. All packets after the initial SYN packet sent by a node should have this flag set.
- PSH – Push flag. If set, asks to push any buffered data to the receiving application.
- RST – Reset flag. If set, resets the connection.
- SYN – Synchronize flag. If set, synchronize sequence numbers. Only the first packet sent from each end should have this flag set.
- FIN – Finished flag – if set, no more data is coming.

Window Size (16 bits) – size of the receive window, which is number of bytes that the receiver is willing to receive.

Checksum (16 bit) – use for error checking of TCP header and data.

Urgent Pointer (16 bits) – if the URG flag is set, this is the offset from the sequence number indicating the last urgent data byte.

Options (from 0 to 10 32-bit words) – optional, not commonly used – see RFC for details.

Protocol Operation

1. Connection is established using a three-way handshake, which creates a virtual circuit.
2. Data is transferred over the virtual circuit until connection is terminated.
3. Connection termination closes the established virtual circuit and releases allocated resources.

TCP operation is controlled by a state machine, with 11 states:

1. LISTEN: wait for connection request from a remote client
2. SYN-SENT: wait for the remote peer to send back a segment with SYN and ACK flags set
3. SYN-RECEIVED: wait for remote peer to send back acknowledgement after sending back a connection
4. ESTABLISHED: port is ready to exchange data with the remote peer
5. FIN-WAIT-1
6. FIN-WAIT-2
7. CLOSE-WAIT
8. CLOSING
9. LAST-ACK
10. TIME-WAIT: insure remote peer has received acknowledgement of termination request (< 4 min.)
11. CLOSED

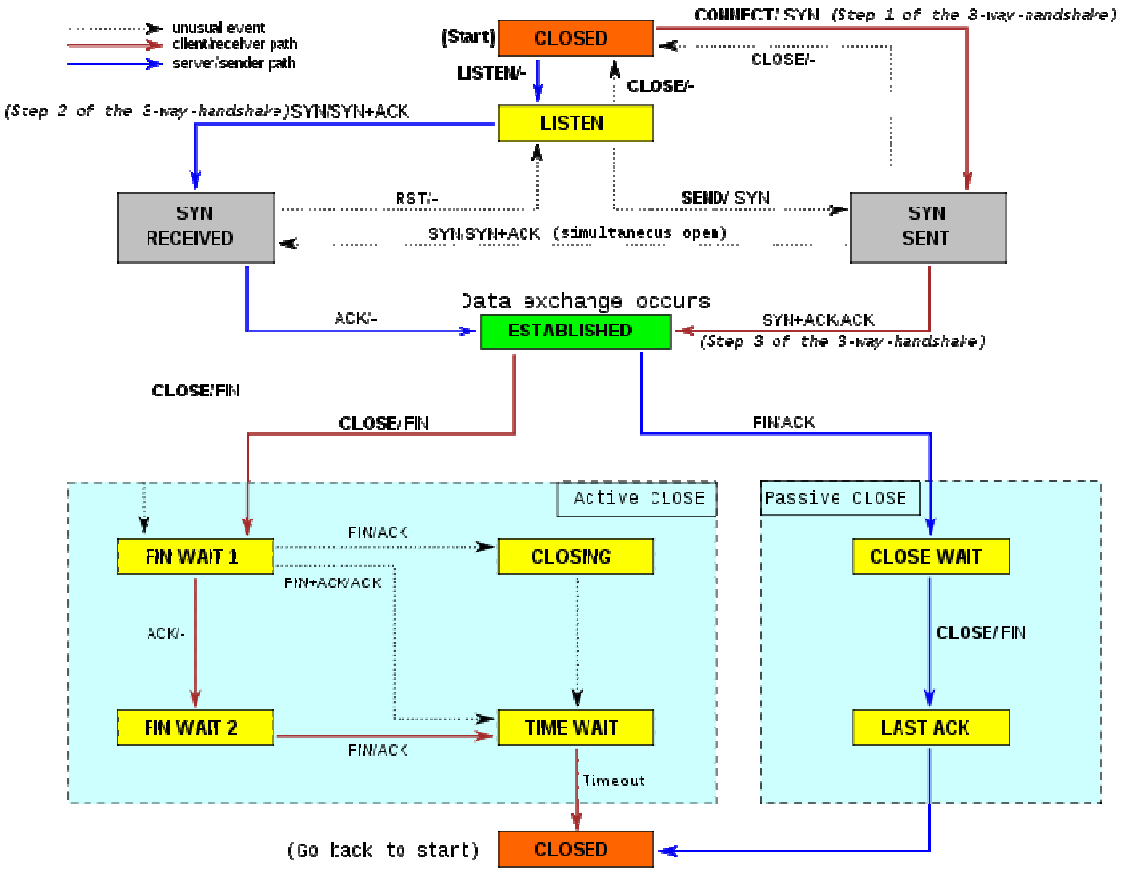


Figure 3.4-c: TCP State Transition Diagram (from Wikipedia)

Notes:

TCP uses sequence numbers to detect lost packets and/or reorder packets that arrive out of order. The cumulative acknowledgment scheme informs the sender that all packets up to the acknowledged sequence number have been received. Selective acknowledgement (RFC 2018) allows for optimization of this feature. Lost data is automatically retransmitted by the sender. End-to-end flow control provides for a mismatch in performance between sender and receiver. A sliding window algorithm allows multiple packets to be in progress which increases efficiency. Recently, congestion control has been added into TCP to avoid network congestion.

3.5 – UDP: The User Datagram Protocol

The User Datagram Protocol is defined in RFC 768, “User Datagram Protocol”, August 1980. Like TCP, it is also a *Transport Layer* protocol. Unlike TCP, UDP implements an *unreliable, connectionless* model. When we say *unreliable* we just mean that error detection and recovery are not built into the protocol, so it is up to the application to do any desired error detection and recovery. By *connectionless*, we mean that each transmission consists of a single (but complete) packet. In TCP/IPv4, a packet is typically 1508

bytes, but can be more or less. If you send a big packet, it will likely be fragmented along the way, and reassembled at the other end. Each datagram is an atomic event, not connected to any other datagram. UDP does not handle *streams* of data (as is done with the connection oriented model). Software that uses UDP does not need to *open* or *close* a connection, it can simply read or write datagrams at any time, and each operation sends or receives one packet. This is a much simpler model than TCP, with less overhead. However, when using UDP you are responsible for doing certain things that TCP does for you, such as error recovery. UDP is often used for things like streaming audio or video. It is also used for DNS queries and responses, and for TFTP (Trivial File Transfer Protocol).

The following standards are relevant to UDP:

- **RFC 768, "User Datagram Protocol", August 1980 (Standards Track)**
- **RFC 2508, "Compressing IP/UDP/RTP Headers for Low-Speed Serial Links", February 1999 (Standards Track)**
- RFC 3095, "Robust header Compression (ROHC): Framework and four profiles: RTP, UDP, ESP and uncompressed", July 2001 (Standards Track)
- RFC 3828, "The Lightweight User Datagram Protocols (UDP-Lite)", July 2004 (Standards Track)
- RFC 4019, "Robust Header Compression (ROHC): Profiles for User Datagram Protocol (UDP) Lite", April 2005 (Standards Track)
- **RFC 4113, "Management Information Base for User Datagram Protocol (UDP)", June 2005 (Standards Track)**
- RFC 4362, "RObust Header Compression (ROHC): A Link-Layer Assisted Profile for IP/UDP/RTP", January 2006 (Standards Track)
- RFC 4727, "Experimental Values in IPv4, IPv6, ICMPv4, ICMPv6, UDP and TCP Headers", November 2006 (Standards Track)
- RFC 4815, "Robust Header Compression (ROHC): Corrections and Clarifications to RFC 3095", February 2007 (Standards Track)
- RFC 5097, "MIB for the UDP-Lite protocol", January 2008 (Standards Track)
- RFC 5225, "RObust Header Compression Version 2 (ROHCv2): Profiles for RTP, UDP, IP, ESP and UDP-Lite", April 2008 (Standards Track)

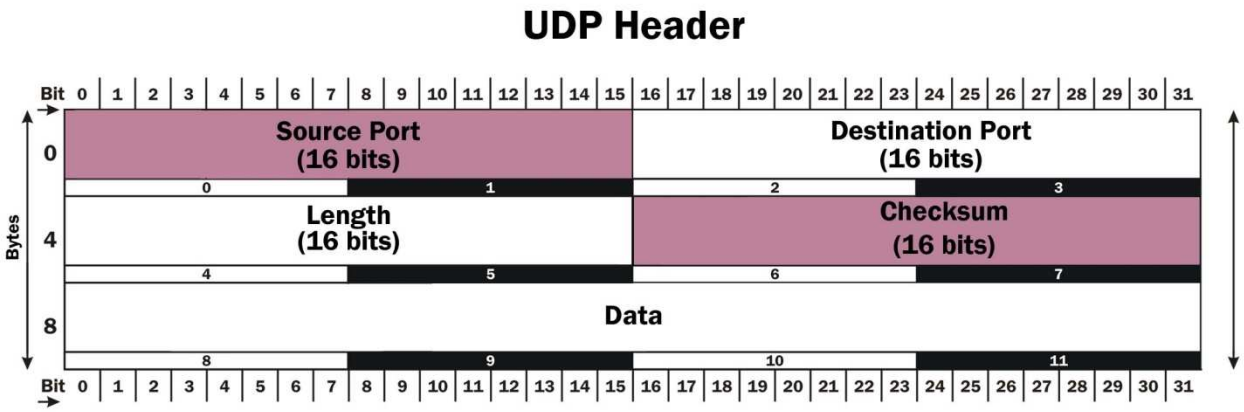


Figure 3.5-a: UDP Packet Header

The *Source Port* field (16 bits) specifies which port number the data is being written to on the sending computer. This field is optional (if not used, fill with zeros).

The *Destination Port* field (16 bits) specifies which port number the data is being read from on the receiving computer.

The *Length* field (16 bits) is the number of bytes in the datagram, including the UDP header and the data, therefore the minimum value is 8 (the length of the UDP header). The maximum value in theory is 65,536 bytes, but this value is limited by the maximum packet size, typically 1508.

The Checksum field (16 bits) is the 16-bit one's complement sum of the 16 bit words in the following items:
1. A "pseudo header" which contains the source and destination IP addresses, the protocol number, and the UDP length (from the IP header).
2. The UDP header itself.
3. The data, padded with a zero byte if required to make an even number of bytes.

The Checksum field is optional (if not used, fill with zeros).

The *Data* field begins immediately after the *Checksum* field. It is not really part of the header, but it is factored into the checksum.

3.6 – DHCPv4: Dynamic Host Configuration Protocol for TCP/IPv4

One of the network services that is really useful in network configuration is the Dynamic Host Configuration Protocol (DHCP). The version that works with IPv4 is now called DHCPv4 (to distinguish it from the one for IPv6, which is called DHCPv6). Without DHCPv4 running on your network, someone must manually configure all network settings on every computer. This can be very complicated and error prone. It also requires at least some expertise which many users don't possess. It is possible to accidently configure two computers with the same address, or mistype a DNS server's address on the 35th computer you install that day. These kinds of errors can cause tricky problems. With a DHCPv4 server, you can configure all the client computers to do "auto configuration". When such a computer powers up, it will search for a DHCPv4 server (or a relay agent, connected to a real DHCPv4 server in another network). When it finds one, it will request configuration data (including the default gateway, the IP addresses of the DNS servers, the Internet domain name, and other items, including a lease on an IPv4 address which should be unique within your network). This makes it easier to change things. If you move a DNS server or make other changes, you need only update your DHCPv4 server configuration, and terminate all client leases (all nodes will request new configuration information).

DHCPv4 is widely used by ISPs, especially ones that have lots more customers than valid (globally routable) IPv4 addresses. They can set very short lease times, then when someone disconnects, the address they had been using can be reused by another customer. Of course these days, most people want 7x24 Internet connectivity, as opposed to perhaps one hour a day or dial-up access. Many ISPs now provide their customers with RFC 1918 private addresses, unless for some reason they specifically require a globally routable address. Some ISPs charge more for a globally routable address, and a *lot* more for multiple globally routable addresses. I have 5 real addresses for my home network, so I can run E-mail and other services, in addition to using one to tunnel IPv6 into my network over IPv4. DHCPv4 can

provide auto configuration with private addresses just as easily as with globally routable addresses, so they still use DHCPv4 to assign those. Basically all of their users are now "hiding" behind a single (or a few) real addresses, via NAT.

DHCPv4 uses broadcast (which doesn't exist in IPv6) and can only deliver 32 bit addresses (for the assigned IP address or things like DNS IP addresses), so it had to be completely rewritten for IPv6. The differences will be covered in the chapter on IPv6.

Most client operating systems in use today (especially on personal computers) include a DHCPv4 client, including all versions of Windows, FreeBSD, Linux, Solaris, Mac OSX, etc. Even smart phones with Wi-Fi include a DHCPv4 client.

Most server operating systems (such as Windows Server, FreeBSD, Linux, etc.) include a DHCPv4 server. The most common one for UNIX and UNIX like servers is *dhcpd* from the Internet Software Consortium (ISC). It is configured by editing some complex ASCII text configuration files (with a text editor). This type of configuration has not changed appreciably in 50 years (and you thought *IPv4* was old). The DHCPv4 server included with Windows Server at least has a GUI configuration tool, which is much easier to use. Most appliances that provide DHCPv4 service include a GUI web based configuration tool (as a "front end" to dhcpd, in most cases).

When you configure a DHCPv4 server, you typically configure one or more pools of addresses to be managed by that server. You cannot have more than one DHCPv4 server in a given network subnet, but DHCPv4 clients cannot contact DHCPv4 servers on another subnet (on the other side of a router). So you either need to have a different DHCPv4 server (or at least a *relay agent*) in every subnet ("broadcast domain"). You can create a "scope" on the server, and configure the "stateless" items that it will use to auto-configure clients, including the domain name, the subnet mask, the address of the default gateway, the IP addresses of two DNS servers, etc. There are dozens of things you can auto-configure with DHCPv4. You also specify a range of addresses (e.g. 192.168.5.100 to 192.168.5.199) as a *pool* from which to lease addresses. You should not manually assign any of these addresses to other nodes. If you do for some reason, you can exclude that address from the available pool.

Once such a server is installed and configured, just set up your client computers to "Obtain an IP address automatically", and to "Obtain DNS server address automatically". As soon as you specify that, or anytime the computer powers up, it will obtain all necessary information (including a unique IPv4 node address) from the DHCPv4 server. In Windows, you can use the "ipconfig /all" command (in a DOS Prompt window) to view the obtained settings (look for the interface named Local Area Connection).

By default, addresses are assigned on a "first come, first served" basis. If you want a given node to be assigned a specific address each time, you can make an *address reservation* by associating one of the pool addresses with that node's MAC address. Any time that node requests configuration data from the DHCPv4 server, it will be assigned that address, rather than a random one from the pool.

3.6.1 – The DHCPv4 Protocol

The DHCPv4 protocol lives in the *Application* Layer. It uses port 67 for data from client to server, and port 68 for data from the server to the client (both over UDP). There are four phases in a DHCPv4 network configuration:

1 IP Discovery
2 IP Lease Offer
3 IP Request
4 IP Lease Acknowledgement

Let's say our network uses 192.168.0.0/16. That means the subnet mask is 255.255.0.0. Our DNS servers are at 192.168.0.11 and 192.168.0.12. The DHCPv4 server is also running on 192.168.0.11. The default gateway is 192.168.0.1. We have created a pool of addresses from 192.168.1.0 to 192.168.1.255.

In the *Discover IP* phase, the client sends a DHCPDISCOVER request, as follows:

- Source Address = 0.0.0.0, Source Port = 68
- Destination Address = 255.255.255.255, Destination Port = 67
- DHCP Option 50: IP address 192.168.1.100 is requested
- DHCP Option 53: Message is DHCPDISCOVER
- Request Subnet mask, Default Gateway, Domain Name and Domain Name Server(s)

In this case, the node is requesting its last known IP address. Assuming it is still connected to the same network, and the address is not already leased to someone else, the server may grant the request. Otherwise the client will have to negotiate for a new address.

In the *DHCP Lease Offer* phase, the server will reserve an IP address for the client (in this case it is accepting the request for the last known address), and send a DHCPOFFER message to the client, as follows:

- Source Address = 192.168.0.11, Source Port = 67
- Destination Address = 255.255.255.255, Destination Port = 68
- DHCP option 01: Subnet Mask is 255.255.0.0
- DHCP option 03: Default gateway is 192.168.0.1
- DHCP option 06: IP addresses of DNS servers are 192.168.0.11 and 192.168.0.12
- DHCP option 51: lease duration is 86400 seconds (1 day)
- DHCP option 53: Message is DHCPOFFER
- DHCP option 54: IP address of DHCP server is 192.168.0.11

In the *IP Request* phase, the client accepts the offer, and sends a DHCPREQUEST message as follows:

- Source Address = 0.0.0.0, Source Port = 68
- Destination Address = 255.255.255.255, Destination Port = 67
- DHCP option 50: IP address 192.168.1.100 is requested
- DHCP option 53: Message is DHCPREQUEST
- DHCP option 54: IP address of DHCP server is 192.168.0.11

In the *IP Acknowledgement* phase, the server officially registers the assignment, and notifies the client of the configuration values:

- Source Address = 192.168.0.11, Source Port = 67
- Destination Address = 255.255.255.255, Destination port = 68
- DHCP option 01: Subnet Mask is 255.255.0.0
- DHCP option 03: Default gateway is 192.168.0.1
- DHCP option 06: IP addresses of DNS servers are 192.168.0.11 and 192.168.0.12
- DHCP option 51: lease duration is 86400 seconds (1 day)
- DHCP option 53: Message is DHCPACK
- DHCP option 54: IP address of DHCP server is 192.168.0.11

At this point, the client actually configures those values for its network interface, and can begin using the network.

3.6.2 – Useful Commands Related to DHCPv4

In Windows, there are some commands available in a DOS prompt box related to DHCPv4:

ipconfig /release	release assigned IPv4 address, deconfigure network
ipconfig /renew	do a new configuration request for IPv4
ipconfig /all	view all network configuration settings

This is an example of the output from "ipconfig /all":

```
C:> ipconfig /all
...
Ethernet adapter Local Area Connection:

   Connection-specific DNS Suffix  . : redwar.org
   Description . . . . . . . . . . . : Realtek PCIe GBE Family Controller
   Physical Address. . . . . . . . . : 00-22-15-24-32-9C
   DHCP Enabled. . . . . . . . . . . : Yes
   IPv4 Address. . . . . . . . . . . : 192.168.1.8(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.0.0
   Default Gateway . . . . . . . . . : 192.168.0.1
   DNS Servers . . . . . . . . . . . : 192.168.0.11
                                       192.168.0.12
   NetBIOS over Tcpip. . . . . . . . : Enabled
...
```

3.7 – TCP/IPv4 Network Configuration

Let's assume our LAN has the following configuration:

Network Address:	192.168.0.0/16	(hence subnet mask = 255.255.0.0)
Default Gateway:	192.168.0.1	

DHCPv4 Address: 192.168.0.11
DNS Server Address: 192.168.0.11, 192.168.0.12
Domain Name: redwar.org

Furthermore, assume the DHCPv4 server is correctly configured with this information, and is managing the address range 192.168.1.0 to 192.168.1.255 (and that some leases have already been granted).

Any node connected to a network with TCP/IPv4 must have certain items configured, including:

- IPv4 node address
- Subnet mask (or equivalently, CIDR subnet mask length)
- IPv4 address of default gateway
- IPv4 addresses of DNS servers
- Nodename
- DNS domain name

3.7.1 – Manual Network Configuration

It is possible perform TCP/IPv4 configuration manually, either by editing ASCII configuration files, as in FreeBSD or Linux; or via GUI configuration tools, as in Windows. If you have understood the material in this chapter, it should be fairly easy to configure your node(s). In most cases, if you have ISP service, the ISP will give you all the information necessary to configure your node(s).

Let's configure a FreeBSD 7.2 node manually. Assign it the nodename "us1.redwar.org", and the IP address 192.168.0.13. The interface we are configuring has the FreeBSD name "vr0".

You need to edit the following files (you will need root privilege to do this):

/etc/rc.conf

```
...
hostname="us1.redwar.org"
ifconfig_vr0="inet 192.168.0.13 netmask 255.255.0.0"
defaultrouter="192.168.0.1"
...
```

/etc/resolv.conf

```
domain      redwar.org
nameserver  192.168.0.11
nameserver  192.168.0.12
```

If you make these changes and reboot, you can check the configuration as shown:

```
$ ifconfig vr0
vr0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500
        options=2808<VLAN_MTU,WOL_UCAST,WOL_MAGIC>
```

```
        ether 00:15:f2:2e:b4:1c
        inet 192.168.0.13 netmask 0xffff0000 broadcast 192.168.255.255
        media: Ethernet autoselect (100baseTX <full-duplex>)
        status: active
$ uname –n
us1.redwar.org
$ nslookup
> server
Default server: 192.168.0.11
Address: 192.168.0.11#53
Default server: 192.168.0.12
Address: 192.168.0.12#53
> exit

$ netstat -rn
Routing tables

Internet:
Destination        Gateway            Flags    Refs      Use  Netif Expire
default            192.168.0.1         UGS         0        5    vr0
...
```

3.7.2 – Auto Network Configuration Using DHCPv4

It is also possible for a node to be automatically configured if a DHCPv4 server (or relay agent) is available somewhere on the LAN (or possibly from the ISP). If you are deploying several nodes on a home network, it is likely that there is a DHCPv4 server in your home gateway / DSL modem.

Let's configure a FreeBSD 7.2 node automatically using DHCPv4. Assign it the nodename "us1.redwar.org", and any IP address from DHCPv4. The interface we are configuring has the FreeBSD name "vr0".

You need to edit the following file (you will need root privilege to do this):

/etc/rc.conf

```
      ...
      hostname="us1.redwar.org"
      ifconfig_vr0="DHCP"
      ...
```

If you make these changes and reboot, you can check the configuration as shown:

```
$ ifconfig vr0
vr0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500
        options=2808<VLAN_MTU,WOL_UCAST,WOL_MAGIC>
        ether 00:15:f2:2e:b4:1c
        inet 192.168.1.9 netmask 0xffff0000 broadcast 192.168.255.255
        media: Ethernet autoselect (100baseTX <full-duplex>)
        status: active
$ uname –n
```

```
us1.redwar.org
$ nslookup
> server
Default server: 192.168.0.11
Address: 192.168.0.11#53
Default server: 192.168.0.12
Address: 192.168.0.12#53
> exit

$ netstat -rn
Routing tables

Internet:
Destination        Gateway            Flags    Refs      Use  Netif Expire
default            192.168.0.1         UGS         0        5    vr0
```

Chapter 4 – The Depletion of the IPv4 Address Space

Many people today are aware that the folks in charge of the Internet are starting to run low on addresses. Most of them are *not* aware that this is not the first time we've faced this, or just how low that pool of addresses is today. The majority of Internet users are either completely oblivious to what is going on and think that the Internet will go on like it has, forever. If they have heard any rumors about an address shortage they have a blind faith that the people in charge can simply work some magic and the problem will go away. Well, they did once, in the mid 1990's (with NAT), but they are all out of tricks this time around. IPv4 is simply out of gas, and it is time to start using its successor, IPv6.

4.1 – OECD IPv6 Report, March 2008

The best study on this done to date (in my opinion) is in the OECD report presented at the OECD Ministerial Meeting on the Future of the Internet Economy, in Seoul Korea, 17-18 June 2008. I was a speaker at the concurrent Korean IPv6 Summit. The full name of OECD is *Organisation for Economic Co-Operation and Development.* It was established in 1961, and currently has 30 member nations, including most members of the EU, plus Australia, Canada, Japan, Korea, Mexico, New Zealand, Turkey, the UK and the US. It had a 2009 budget of EUR 320 million. Their goals are to:

- Support sustainable economic growth
- Boost employment
- Raise living standards
- Maintain financial stability
- Assist other countries' economic development
- Contribute to growth in world trade

Unlike the IETF or ISO, the OECD is not specifically concerned with technology. They are primarily concerned with the economies of their member countries. However, they have determined that the imminent exhaustion of the IPv4 address space will have a major impact on most of their goal areas. Because of this, they did a major study, the results of which are presented in Ministerial Background Report DSTI/ICCP(2007)20/FINAL, "Internet Address Space: Economic Considerations in the Management of IPv4 and in the Deployment of IPv6". The report is available free by download over the Internet (search for "OECD IPv6 Report"). You should actually read the entire report, but I will summarize the most important aspects of it in this chapter.

Let me quote one paragraph from the Main Points section:

"There is now an expectation among some experts that the currently used version of the Internet Protocol, IPv4, will run out of previously unallocated address space in 2010 or 2011, as only 16% of the total IPv4 address space remains unallocated in early 2008. The situation is critical for the future of the Internet economy because all new users connecting to the Internet, and all businesses that require IP

addresses for their growth, will be affected by the change from the current status of ready availability of unallocated IPv4 addresses."

As of this writing, in early 2010, only 8% of the addresses remain unallocated. The current best estimates are that the IANA address pool will be exhausted by July 2011, and all RIRs will exhaust their supply within six months after that (some potentially even earlier).

Another key passage from this section follows:

"As the pool of unallocated IPv4 addresses dwindles and transition to IPv6 gathers momentum, all stakeholders should anticipate the impacts of the transition period and plan accordingly. With regard to the depletion of the unallocated IPv4 address space, the most important message may be that there is no complete solution and that no option will meet all expectations. While the Internet technical community discusses optional mechanisms to manage IPv4 address space exhaustion and IPv6 deployment and to manage routing table growth pre- and post- exhaustion, governments should encourage all stakeholders to support a smooth transition to IPv6."

"IPv6 adoption is a multi-year, complex integration process that impacts all sectors of the economy. In addition, a long period of co-existence between IPv4 and IPv6 is projected during which maintaining operations and interoperability at the application level will be critical. The fact that each player is capable of addressing only part of the issue associated with the Internet-wide transition to IPv6 underscores the need for awareness raising and co-operation".

Basically, there is no solution for those wanting to remain with IPv4. It is going to take *multiple years* to make the transition. There are only two years left, so March 2010 is really the last possible date to begin a smooth and affordable introduction of IPv6. Any later start will involve unnecessary expense and crisis management, towards the end of the IPv4 lifetime. Such transitions are usually not done well when rushed. And once the addresses are gone, that's it.

The report acknowledges that in the early phases of a major technology transition such as this, there may be little or no incentive to shift to the new technology. However, once a critical mass of users adopt the new technology, there is often a *tipping point* after which adoption grows rapidly until it is widespread. In theory this tipping point is reached when the marginal cost, for an ISP or an organization, of implementing the next device with IPv4 becomes higher than the cost of deploying the next device with IPv6. For an ISP, there are costs associated with deploying IPv4 nodes such as the cost of obtaining the addresses themselves, the costs of designing and deploying network infrastructure that uses fewer and fewer public (globally routable) addresses (by using NAT). When these become higher than the cost of deploying IPv6, they will begin migration in earnest. Reaching this tipping point depends on a number of factors, including customer demand, opportunity costs, emerging markets, the introduction of new services, government incentives, and regulation.

One of the key requirements for migrating to IPv6 is technical expertise in the subject. This is necessary to provide economies and companies with competitive advantage in the area of technology products and services, and the benefit from ICT-enabled innovation. Countries who are early adopters, and provide training and incentives for their companies to embrace it, or even help fund the necessary infrastructure (as in China) will have significant competitive advantages in years to come over countries that are laggards in this transition.

Increasing scarcity of IPv4 addresses can raise competitive concerns in terms of barriers to new entry and strengthening incumbent positions. There has been much discussion over how to manage previously allocated IPv4 addresses once the free pool has been exhausted. Will a black (or even a legitmate) market evolve for IPv4 addresses? Will companies that have more than they need be selling them on eBay? It's possible that some companies might even be acquired in order to obtain a large number of addresses (as happened when Compaq bought Digital Equipment Corporation, and then again when HP bought Compaq). Today, you only *borrow* (lease?) addresses from an ISP for so long as you have service with that ISP. If you terminate that service, the addresses are reclaimed by the ISP for allocation to other customers. You don't really *own* those addresses, so you can't *sell* them. Even the ISP doesn't own them, if an ISP goes out of business their address pool probably returns to the RIR they got them from. Some of these situations are not currently well defined, but they will be as the IPv4 address space nears exhaustion. Notably, the situation on the early Class A block allocations is not quite so well defined. Those blocks *may* be owned by those early adopter companies.

There is also discussion of how existing and increasing use of NAT requires developers of network aware products and applications to build increasingly complex central gateways or NAT traversal mechanisms to allow clients (who are in most cases *both* behind NAT gateways) to communicate. This is creating barriers to innovation and to the development of new services. It is also causing problems with the overall performance and stability of the Internet.

There is a risk of some parts of the world deploying IPv6, while others continue running IPv4 with multiple layers of NAT. Such decisions would impact the economic opportunities offered by the Internet with severe repercussions in terms of stifled creativity and deployment of generally accessible new services. Also, there could be serious issues of interoperation between people in the IPv6 world and those left behind in the IPv4 world. This could lead to a fragmentation of the Internet.

The five sections of the report cover the following topics:

- Overview of the major initiatives that have taken place in Internet addressing to-date, and the parallel development of institutions that manage Internet addressing.
- Summary of proposals under consideration for management of remaining IPv4 addresses.
- Overview of the drivers and challenges for transitioning to IPv6 through a dual stack (IPv4 + IPv6) environment. It reviews factors that influence IPv6 adoption, drawing on available information.
- Economic and public policy considerations and recommendations to governments.
- Lessons learned from several IPv6 deployments.

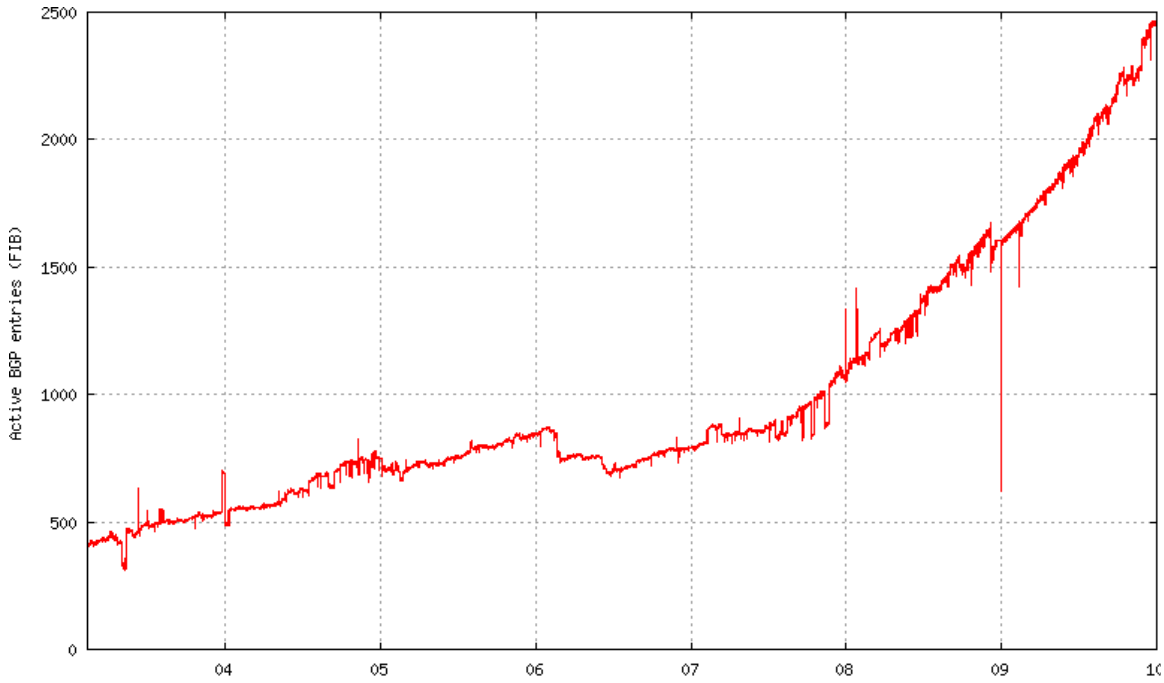
4.2 – OECD Follow-up Report, April 2010

In April 2010, the OECD released a follow-up report to the IPv6 report mentioned above. It is called "Internet Addressing: Measuring Deployment of IPv6". They still expect IPv4 addresses to run out in 2012. As of March 2010, only 8% of the full IPv4 address space is available for allocation. Currently, IPv6 use is growing faster than IPv4 use, albeit from a still small base. Several large-scale deployments are taking place or are in planning. Some of the key findings, all as of March 2010 are:

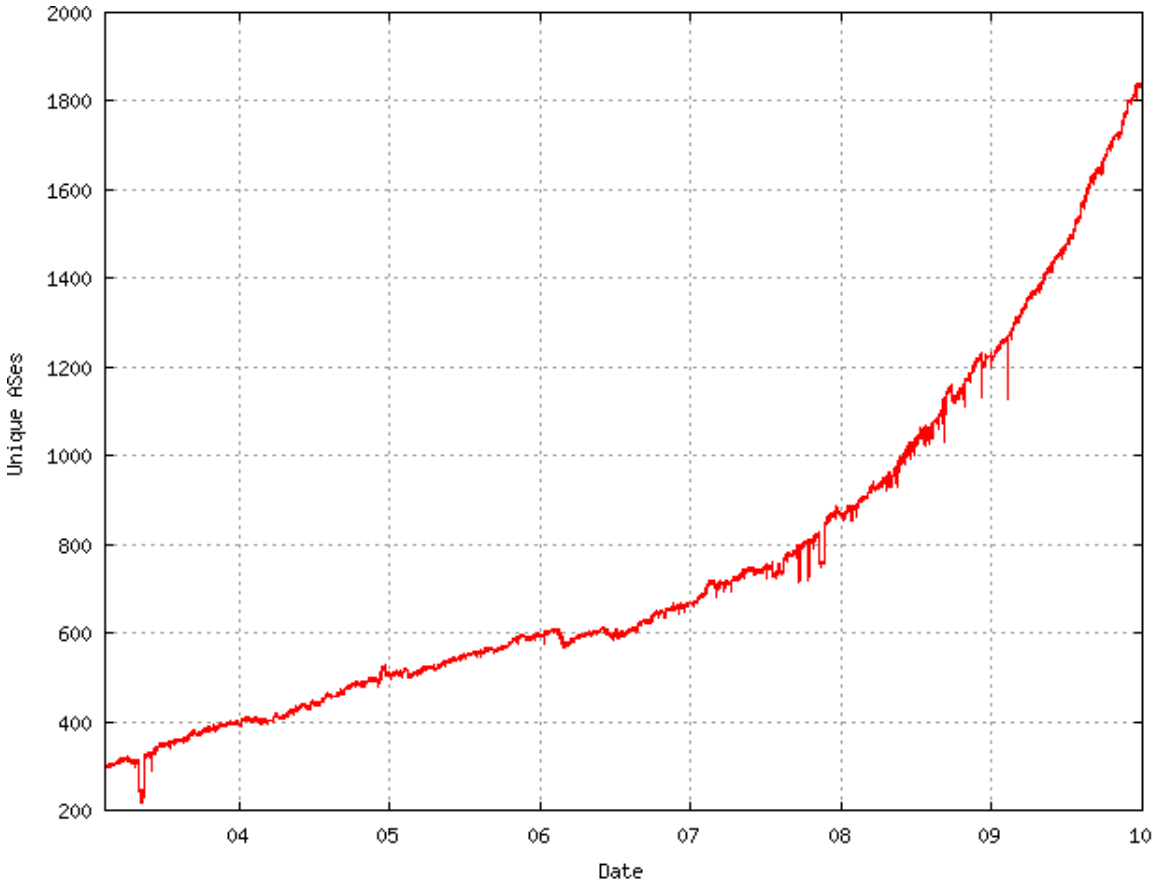
- 5.5% of the networks on the Internet (1,800 networks) can handle IPv6 traffic.
- IPv6 networks have grown faster than IPv4-only networks since mid-2007.
- Demand for IPv6 address blocks has grown faster than demand for IPv4 address blocks.
- One out of five transit networks (i.e. networks that provide connections through themselves to other networks) handle IPv6. This means that Internet infrastructure players are actively readying for IPv6.
- As of January 2010, over 90% of installed operating systems are IPv6 capable, and 25% of end users ran an operating system that enabled IPv6 by default (e.g. Windows Vista or Mac OS X). This percentage has probably increased since the release of Windows 7, but no measurement is available.
- As of January 2010, over 1.45% of the top 1000 websites were available over IPv6, but as of March 2010 (when Google IPv6 enabled their websites) this jumped to 8%.
- Over 4,000 IPv6 prefixes (address blocks) had been allocated. Of these 2,500 (60%) showed up as routed on the Internet backbone (were actually in use).
- At least 23% of Internet eXchange Points explicitly supported IPv6.
- 7 out of 13 DNS Root Servers are accessible over IPv6.
- 65% of Top Level Domains (TLDs) had IPv6 records in the root zone file.
- 80% of TLDs have name servers with an IPv6 address.
- 1.5 million domain names (about 1% of the total) had IPv6 DNS records.

Operators in the RIPE NCC and APNIC service areas were given a survey in 2009. The results showed:

- 7% of APNIC respondents claimed to have equal or more IPv6 traffic than IPv4 traffic.
- 2% of RIPE respondents claimed to have equal or more IPv6 traffic than IPv4 traffic.
- Of those respondents not deploying IPv6, 60% saw cost as a major barrier.
- Of those respondents deploying IPv6, 40% considered lack of vendor support the main obstacle.



Routed IPv6 Prefixes, 2003 to end of 2009



IPv6 unique Autonomous Systems, 2003 to end of 2009

Source: ITAC/NRO Contribution to the OECD, Geoff Huston and George Michaelson, data from end of year 2009.

Since 2008, the ratio of routed IPv6 prefixes to IPv4 prefixes has climbed from 0.45% to 0.8%, which indicates that the number of routed IPv6 prefixes is increasing more rapidly than that of routed IPv4 prefixes. The ratio of IPv6 to IPv4 AS entities actively routing went from about 3.2% in 2008 to 5.5% in 2010.

The compound annual growth rate from 24 February 2009 to 5 November 2009 for dual stack ASes was 52%, for IPv6-only ASes was 13%, and for IPv4-only ASes was 8%. At year end 2009, there were 31,582 ASes using IPv4-only, there were 1806 ASes using dual stack, and there were 59 ASes using IPv6-only.

One trend is that service providers, corporations, public agencies and end-users are using IPv6 for advanced and innovative activities on private networks. IPv6 is also being used in 6LowPAN (IPv6 over Low power Personal Area Networks), as specified in RFC 4944, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", September 2007.

4.3 – How IPv4 Addresses Were Allocated in the Early Days

In the early days, before IANA and the RIRs were created, IPv4 addresses were actually allocated manually by a single individual, Jon Postel. He never dreamed how large the Internet would grow, or that it would be a worldwide phenomenon that had a major impact on most world economies. He is the one responsible for allocating large chunks ("Class A" blocks) to a few early adopters (e.g. HP, Apple, and M.I.T.) Unfortunately, those allocations are very difficult to undo today, so about 1/3 of all the addresses allocated in the U.S. belong to less than 50 organizations. The IANA now just considers those *legacy allocations*, and has tried to do the best they could with the address space remaining at the time they took over allocation.

4.3.1 – Original "Classful" Allocation Blocks

The first 50% of the full IPv4 address space (0.0.0.0 to 127.255.255.255) was divided up into 128 "Class A" blocks (now known as "/8" or "slash-8" blocks). Each of these contained 2^{24}-2, or some 16.8 million usable addresses. Here is a list of some of the lucky organizations that own these blocks today, either from the original allocation or by buying other companies that owned them.

General Electric	3.x.x.x	U.S. DoD DISA	22.x.x.x
Level 3 Communications	4.x.x.x	U.K. Ministry of Defense	25.x.x.x
U.S. Army Info Systems Center	6.x.x.x	U.S. DoD DISA	26.x.x.x
(formerly DoD, now ARIN)	7.x.x.x	U.S. DoD DSI-North	28.x.x.x
Level 3 Communications	8.x.x.x	U.S. DoD DISA	29.x.x.x
IBM	9.x.x.x	U.S. DoD DISA	30.x.x.x
U.S. DoD Intel Info Systems	11.x.x.x	AT&T Global Network Svcs	32.x.x.x

AT&T Worldnet	12.x.x.x	U.S. DoD DLA Sys Auto Ctr	33.x.x.x
Xerox Corp.	13.x.x.x	Halliburton	34.x.x.x
HP	15.x.x.x	InterOp Show	45.x.x.x
DEC (now HP)	16.x.x.x	Bell-Northern (now Nortel)	47.x.x.x
Apple Computer	17.x.x.x	Prudential Insurance	48.x.x.x
Mass. Inst. Of Technology	18.x.x.x	E.I. DuPont de Nemours	52.x.x.x
Ford Motor Company	19.x.x.x	Daimler AG	53.x.x.x
Computer Sciences Corp	20.x.x.x	U.S. DoD Network Info Ctr	55.x.x.x
DoD DISA DDN-RVN	21.x.x.x	U.S. Postal Service	56.x.x.x

Another 25% of the full address space (128.0.0.0 to 191.255.255.255) was divided up into 16,384 "Class B" blocks (now known as "/16" blocks). Each of these contained 2^{16}-2, or 65,534 usable addresses.

Another 12.5% of the full address space (192.0.0.0 to 224.255.255.255) was divided up into about 2.1 million "Class C" blocks (now known as "/24" blocks). Each of these contained 2^{8}-2, or 254 usable addresses.

Another 6.25% of the full address space (224.0.0.0 to 239.255.255.255) was reserved for multicast (these are known as Class D addresses). There is no way to "recover" any of this address space.

The final 6.25% of the full address space (240.0.0.0 to 255.255.255.255) was reserved for future use, experimentation and limited broadcast. These are known as Class E addresses. These addresses cannot be "recovered" without modifications to essentially every router in the world (most routers block them by default – in many routers this is not even configurable).

The subblock of Class E from 255.0.0.0 to 255.255.255.255 is actually used for "limited broadcast" (limited because it will not cross routers). A packet sent to any of these addresses will be received by all nodes on your LAN. Of these, normally only the address 255.255.255.255 is used. There is no broadcast in IPv6 (although there is a multicast address that has much the same effect).

The U.S. Deptartment of Defense has 10 "/8" blocks, for about 168 million addresses. This is almost 4% of the total IPv4 address space. One entire "/8" block (127.x.x.x) has only one address used, which is 127.0.0.1 (the IPv4 "loopback" address, used to address your own node). A small block at 169.254.0.0/16 is reserved for IPv4 Link Local usage (similar to IPv6 link-local addresses). For details, see RFC 5735, "Special Use IPv4 Addresses", January 2010.

One "/8" block (10.0.0.0/8), one "/12" block (172.16.0.0/12) and one "/16" block (192.168.0.0/16) were reserved for use as "private" addresses by RFC 1918, "Address Allocation for Private Internets", February 1996. These addresses can be used by any organization for any internal network, but should never be routed onto the Internet (although in practice you can sometimes find these addresses on the backbone due to misconfigured routers). These would correspond to internal phone "extensions" such as 101, 102, etc. Every company with a PBX might use that same set of extensions.

As of 4 June 2010, only 16 of the possible 256 "/8" blocks (about 6.25% of the full address space) are still unallocated. Here is a map of the status of all 256 "/8" blocks. By September 2011, (or earlier) there won't be any dots left. All the blocks with dots (unallocated "/8"s) in the chart today will be allocated to one of the RIRs (ARIN, RIPE, APNIC, LACNIC or AfriNIC).

```
      +0    +1    +2    +3    +4    +5    +6    +7    +8    +9
000   R     AP    RN    L     L     .     L     L-AR  L     L
010   R     L     L     L     AP    L     L     L     L     L
020   L     L     L     .     AR    L     L     AP    L     L
030   L     RN    L     L     L     L     .     .     L     .
040   L     AF    .     L-AP  L     L     RN    L     L     .
050   AR    L     L     L     L     L     L     L     AP    AP
060   AP    AP    RN    AR    AR    AR    AR    AR    AR    AR
070   AR    AR    AR    AR    AR    AR    AR    RN    RN    RN
080   RN    RN    RN    RN    RN    RN    RN    RN    RN    RN
090   RN    RN    RN    RN    RN    RN    AR    AR    AR    AR
100   .     .     .     .     .     .     .     AR    AR    RN
110   AP    AP    AP    AP    AP    AP    AP    AP    AP    AP
120   AP    AP    AP    AP    AP    AP    AP    R     L-AR  L-AR
130   L-AR  L-AR  L-AR  L-AP  L-AR  L-AR  L-AR  L-AR  L-AR  L-AR
140   L-AR  L-RN  L-AR  L-AR  L-AR  L-RN  L-AR  L-AR  L-AR  L-AR
150   L-AP  L-RN  L-AR  L-AP  L-AF  L-AR  L-AR  L-AR  L-AR  L-AR
160   L-AR  L-AR  L-AR  L-AP  L-AR  L-AR  L-AR  L-AR  L-AR  L-AR
170   L-AR  L-AP  L-AR  AR    AR    AP    RN    LA    RN    .
180   AP    LA    AP    AP    AR    .     LA    LA    L-RN  LA
190   LA    L-LA  L-AR  RN    RN    RN    L-AF  AF    L-AR  AR
200   LA    LA    AP    AP    AR    AR    AR    AR    AR    AR
210   AP    AP    RN    RN    L     L     AR    RN    AP    AP
220   AP    AP    AP    AP    R-MC  R-MC  R-MC  R-MC  R-MC  R-MC
230   R-MC  R-MC  R-MC  R-MC  R-MC  R-MC  R-MC  R-MC  R-MC  R-MC
240   R-FU  R-FU  R-FU  R-FU  R-FU  R-FU  R-FU  R-FU  R-FU  R-FU
250   R-FU  R-FU  R-FU  R-FU  R-FU  R-FU
```

Key and Analysis

AR	ARIN allocated	33	72	28.13%	(ARIN total)
L-AR	Legacy, admin by ARIN	39			
AP	APNIC allocated	38	44	17.19%	(APNIC total)
L-AP	Legacy, admin by APNIC	6			
RN	RIPE NCC allocated	33	37	14.45%	(RIPE total)
L-RN	Legacy, admin by RIPE NCC	4			
LA	LACNIC allocated	8	9	3.52%	(LACNIC total)
L-LA	Legacy, admin by LACNIC	1			
AF	AfriNIC allocated	2	4	1.56%	(AfriNIC total)
L-AF	Legacy, admin by AfriNIC	2			
L	Legacy, early allocation	39	39	15.23%	(Legacy total)
R	Reserved	3	3	1.17%	
R-MC	Reserved, Multicast	16	16	6.25%	
R-FU	Reserved, Future Use	16	16	6.25%	
.	Unallocated	16	16	6.25%	(Unallocated)
		256	256	100.00%	

Almost all of the “Legacy, early allocation” blocks are in the U.S., so ARIN’s real share of the total IPv4 address space is over 40% (for less than 5% of the world’s population).

You can check the official status of the remaining allocations at any time, at:

http://www.iana.org/assignments/ipv4-address-space/ipv4-address-space.xml

4.3.2 – Classless Inter-Domain Routing (CIDR)

The original allocation block sizes (Classes A, B & C) did not fit all organizations. For many organizations, even the smallest block (Class C) was too big. If we had stuck with the original allocation block sizes, we would have run out of addresses around 1997. When this was realized, the IETF introduced Classless Inter-Domain Routing as defined in RFC 1518, "An Architecture for IP Address Allocation with CIDR", September 1993; and RFC 1519 "Classless Inter-Domain Routing (CIDR): An Address Assignment and Aggregation Strategy", September 1993. CIDR allowed the two parts of an address to be split along any of the 30 possible places to divide them, not just at multiples of 8 bits. Some useful CIDR allocation block sizes are:

Size	Subnet Mask	Number of usable addresses in block
/30	255.255.255.252	2
/29	255.255.255.248	6
/28	255.255.255.240	14
/27	255.255.255.224	30
/26	255.255.255.192	62
/25	255.255.255.128	126
/24	255.255.255.0	254 (old Class C)
/23	255.254.0.0	510
/22	255.252.0.0	1,022
/21	255.248.0.0	2,046
/20	255.240.0.0	4,094
/19	255.224.0.0	8,190
/18	255.192.0.0	16,382
/17	255.128.0.0	32,766
/16	255.255.0.0	65,534 (old Class B)
/8	255.0.0.0	16,777,214 (old Class A)

CIDR allows a closer fit to actual organization size than the old classful "3 sizes fit all" scheme. However, each allocated block requires an entry in the core routing tables. As we allocate smaller and smaller blocks, the number of entries in the core routing tables is growing *very* rapidly. Many things are beginning to go wrong as we get closer and closer to an empty barrel.

In the mid 1990's, there were steps taken (NAT and Private Addresses) to further limit the number of addresses being allocated to each organization. NAT was only ever envisioned by its creators as a "quick fix" that would buy us a few years to really solve the problem. They understood all the problems NAT would cause, and were willing to live with them for a short time, when the alternative was to run out of IPv4 addresses somewhere around 1997. For the real long-term fix, the IETF also began working on the next generation Internet Protocol with a much larger address space. That next generation Internet Protocol is complete, mature and available to deploy today. It is called IPv6.

4.4 – Problems Introduced by Customer Premise Equipment NAT (CPE NAT)

Since the mid 1990's we have been living with problems created by the introduction of Network Address Translation doing conventional "hide mode" (Cone) NAT at the Customer Premise (CPE NAT). These include:

- Difficulty for internal nodes to accept incoming connections, for VoIP (SIP), Peer-to-Peer (P2P), running your own mail (SMTP), web (HTTP/HTTPS), File Transfer (FTP/SSH) or other servers.

- Problems with protocols that embed IPv4 addresses in packet transmissions (SIP, many games).

- Problems with protocols that detect tampering to IP and/or TCP/UDP header fields (e.g. IP addresses, port numbers), such as IPsec Authentication Header (AH).

- Problems due to advances in web technology (primarily Web 2.0 / AJAX) that use large numbers of connections, each over a different port, such as iTunes and Google Maps. This can be as high as 200 ports per application. Since NAT systems share the 65,536 possible ports associated with a single "real" IPv4 address among the nodes hidden behind each address, each internal user on average can use at most 65,536 divided by the number of users behind that address. In enterprise networks, this might (until recently) have been thousands or tens of thousands of nodes behind one real address. For 1,000 nodes, on average each user could use no more than 65 ports. For 10,000 nodes, on average each user could use no more than 6 ports. To allow each user up to 200 ports, no more than 300 users should be hidden behind each IPv4 address. Currently, the *average* number of ports used per user is actually quite low (less than 10), but this is expected to grow rapidly as more users begin using Web 2.0 / AJAX type applications. If possible, NAT schemes should use ports on a *first come, first served* basis, rather than allocating 1/n of the possible ports to each node.

- Difficulty of tracking abuse to specific users behind a NAT. This requires keeping large amounts of information including source IP address, destination IP address, port number(s) and accurate time stamps for *every* connection. This may have to be kept for up to one year. A year's worth of such data for a single user can be tens of gigabytes to terabytes in size. Multiplied by the number of users, this is a staggering amount of storage that ISPs are required to keep. Hackers love to "hide behind" NAT gateways.

Essentially, private IP addresses behind "hide mode" NAT are good only for outgoing connections using the simplest connectivity paradigms (e.g. client to server, using a small total number of ports per user).

It is possible to allow *at most* one internal node to accept incoming connections on a given port (e.g. port 80 for HTTP) for the gateway external IPv4 address, using *port forwarding*. For example, your NAT gateway can be configured to forward any incoming connection to its external IPv4 address on port 25, to the private address of a single internal node where an E-mail server is running. The gateway could *also* forward incoming connections to its external IPv4 address on port 80 to the same (or a different) internal node's private address where a web server is running. This limits the entire LAN (or that part of it behind a given real IPv4 address) to a single server for any given port number (when using port redirection). This still translates the destination IPv4 address on the way in, and the source IPv4 address on the way out (but not port numbers are left unchanged). This still causes many of the problems listed above. One-to-one NAT (BINAT) does not have this limitation, but one valid external IPv4 address (in

addition to the valid external IPv4 address used for hide-mode NAT and port redirection) is required for each internal server.

If you tried to map incoming port 80 traffic to *two* different internal addresses with port redirection, your browser would be very confused by receiving responses from two different web servers simultaneously. A good firewall or router should flag the attempt to do this as an error.

Some firewalls (or other NAT gateways) in addition to "hide mode" (Cone) NAT for outgoing connections, and port forwarding, also support *bidirectional NAT* (called BINAT, symmetric NAT, and "1 to 1" NAT among other names). This type of NAT makes a two way address translation between a single external IP address and a single private internal address (hence "1 to 1"). The full 65,536 possible ports may be used on the internal node, but a distinct real IPv4 address is required for *each* such BINAT mapping. This would allow deployment of multiple web servers within a LAN, or an easy way to provide access to many services on a single node (e.g. a Windows Server based computer). This still translates the destination IP address of packets on the way in, and the source IP address of packets on the way out (again, port numbers are not affected) still causing many of the problems listed above. In addition it uses up one real address per internal server, and requires addressing the "missing ARP" problem (caused by the fact that there is no physical node at the external address to respond to ARP queries). This can be solved by configuring a static ARP for the external IPv4 address on the NAT gateway, or various other solutions. Solving the "missing ARP" problem is one of the most difficult and least widely understood aspects of managing a NAT gateway (or firewall).

There is in fact an external interface on the NAT gateway, with a valid external address (say 123.45.67.81), which will reply as usual to an ARP request to its *primary* address (e.g. 123.45.67.81) with the MAC address of the external interface. With BINAT, however, you also assign an additional *alias* address for each BINAT mapping (e.g. 123.45.67.82, 123.45.67.83, etc.) to the *gateway's* external interface. If an ARP is done for to of these *alias addresses*, by default the external interface will not respond to them, hence the "missing ARP" (actually, "missing ARP *response*"). The ARP request is not translated to the internal node, and even if it was, the node doing the ARP doesn't want the MAC address of the internal node, it wants the MAC address of the external interface of the gateway. To get the external interface to respond with its MAC address to ARP requests for an *alias* address, you must configure a *proxy ARP* on the external interface *for that alias address*. The commands for configuring alias addresses on the external interface, and proxy ARPs for them, vary widely from one OS to another. See the labs in Chapter 10 for an example of this with m0n0wall (based on FreeBSD). In some cases, other mechanisms may be used to solve the "missing ARP" problem, such as configuring a static route for each alias address. This eliminates the need for other nodes to do an ARP request.

There are several *NAT Traversal* protocols (STUN, TURN, SOCKS, NAT-T, etc.) that allow incoming connections to internal nodes that have only private addresses (without any port forwarding or BINAT support in the NAT gateway). These typically require an outside server to assist (this alone should raise security and reliability concerns). STUN uses an outside server only to *establish* the connection, while TURN also routes all traversing traffic through an outside gateway. All NAT traversal schemes involve encapsulating traffic over UDP, which complicates error detection & recovery and intrusion detection, as well as supporting the "connection oriented" nature of TCP traffic. All require extensive modifications to the source code of clients, which is quite complex and very specific to the NAT traversal algorithm used. Usually the external servers used are not under control of the network, leading to security issues. One of the most popular network applications (Skype) uses standard UDP encapsulated "hole punching"

traversal, which causes *many* security issues Anyone with access to the external server can easily track who you are calling (and who is calling you), and even *listen in* or redirect the call. With IPv6, there is no NAT, hence no need for NAT traversal.

Note that there are many variants of NAT, and a given implementation of NAT traversal may work with only one or two of them. Also, many schemes fail if there are two or more NAT mappings in series (say your ISP doing a NAT44 mapping to one private address, then your CPE router/modem doing a *second* NAT44 mapping of that private address to yet another private address (this is sometimes called NAT444).

RFCs related to NAT Traversal

- **RFC 1928, "SOCKS Protocol Version 5", March 1996 (Standards Track)**
- RFC 3489, "Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)", March 2003 (Standards Track, Obsoleted by RFC 5389)
- **RFC 3947, "Negotiation of NAT-Traversal in the IKE", January 2005 (Standards Track)**
- **RFC 3948, "UDP Encapsulation of IPsec ESP Packets", January 2005 (Standards Track)**
- **RFC 5389, "Session Traversal Utilities for NAT (STUN)", October 2008 (Standards Track)**
- **RFC 5766, "Traversal Using Relays Around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)", March 2010 (Standards Track, awaiting final approval)**

4.5 – Implementing NAT at the Carrier: Carrier Grade NAT (CGN) or Large Scale NAT (LSN)

As we progress from the "end times" for IPv4, to "life *after* IPv4" (beyond the depletion date for IPv4), those who have not already migrated to IPv6 will face even greater problems, as ISPs deploy *Carrier Grade NAT* or *Large Scale NAT* solutions in their networks, as opposed to doing it in the Customer Premise Equipment (CPE NAT). The reason for this is to try to make optimal use of an even smaller number of globally routable IPv4 addresses than is possible with CPE NAT. Essentially the ISP will have a very small pool of real IPv4 address (less than the number of customers). They will share single real IPv4 addresses *across customers*. This will make the problems associated with CPE NAT dramatically worse. There is excellent coverage of the issues associated with deploying NAT in the carrier in *draft-ford-shared-addressing-issues-02* (2010-03-08). The schemes discussed in this Internet Draft include:

- Dual-Stack Lite, *draft-ietf-softwire-dual-stack-lite*
- Carrier Grade NAT (CGN), *draft.nishitani-cgn*
- NAT64, *draft-ietf-behave-v6v4-xlate-stateful*
- IVI, *draft-ietf-behave-v6v4-xlate*
- Address+Port (A+P) proposals, *draft-ymbk-aplusp*, *draft-boucadair-port-range*
- Scalable Multihoming across IPv6 – Stateless Address Mapping, *draft-despres-sam*

Of these, only Dual-Stack Lite makes dual stack service available to users. It provides direct IPv6 service (no NAT, no tunneling). It provides IPv4 service tunneled over IPv6 (called 4in6 tunneling) with only one level of NAT44 (which takes place at the carrier). Customers will get *only* private IPv4 addresses. It is possible that some ISPs may provide a few precious "real" (globally routable) IPv4 addresses to business customers at a significant price premium (*all the market will bear*, which could easily reach thousands of

dollars per address per year). All of the NAT schemes extend the address space by adding port information. They differ in the way they manage the port value.

With CPE NAT, a given real IPv4 address covered only one legal entity (a home, a company, etc.). With carrier based NAT, multiple legal entities will be behind most real IPv4 addresses, which will vastly complicate the legal issues (such as tracking down a source of network abuse or being able to prove who really did something).

You will see the terms *NAT444* and *NAT464* in discussions of carrier based NAT. The existing NAT that is widely deployed now called is *NAT44* (NAT from IPv4 to IPv4). There is also *NAT46* (NAT from IPv4 to IPv6 and *NAT64* (NAT from IPv6 to IPv4).

NAT444 essentially leaves the CPE NAT44 (the existing one layer NAT that is widely deployed today) intact at the Customer Premise, while the carrier deploys a *second* layer of NAT44 before it ever reaches the customer. It is really just two NAT44 mechanisms in series. The CPE NAT44 will map the private addresses supplied from the carrier NAT44 onto yet another set of internal private addresses. The transport from carrier to customer is also over IPv4. The difference from existing systems is that today the CPE NAT usually has one real IPv4 address which it shares among multiple internal nodes. In NAT444 systems, there won't be *even one* real IPv4 address at the customer premise. It will be quite difficult (and probably very expensive) to host servers with public IPv4 addresses (e.g. web, mail, VoIP) at customer sites – most will have to be hosted at a collocation facility.

NAT464 is similar, but involves doing one layer of NAT46 (from IPv4 to IPv6) at the carrier, followed by a second layer of NAT64 (from IPv6 to IPv4) at the customer premise. This allows the transport from carrier to customer to be over IPv6, which is a good thing, but involves upgrading or replacing all Customer Premise equipment to ones that are NAT64 compliant (few are today). Also, address translation between IP families (IPv4 to IPv6 and IPv6 to IPv4) has even more problems than address translation within a single IP family (only IPv4 to IPv4 – *there is no IPv6 to IPv6 NAT*!).

For an analogy, imagine deploying *nested* telephone PBXes. There would be an outer PBX, with a real telephone number, and behind that other PBXes with internal extensions from the outer PBX. Behind each internal PBX, you would have sets of internal phones. To call an internal phone, you would dial the real phone number of the outer PBX, have to do something to select an internal PBX (dial the internal PBX's extension number?) then once connected to the internal PBX, you would need to interact with *it* to select an internal phone (e.g. dial the first three characters of the phone owner's name). This is the kind of complexity that IPv4 applications will now have to cope with. It will be much simpler to just convert them directly to IPv6.

In either case (NAT444 or NAT464), there are some protocols that will work across one layer of NAT, but fail when there is more than one layer of NAT. Both NAT444 and NAT464 will introduce these kinds of issues, since both involve *at least* two layers of NAT. Some home or small business users may unintentionally introduce *even more* layers of NAT due to lack of understanding, for example by deploying a firewall/NAT box behind a modem/NAT gateway.

The following problems are made worse by Carrier Grade NAT compared even to CPE NAT. Some affect only the end user, some affect third parties (e.g. law enforcement), and many affect both.

- The number of ports available per node will be even less, so Web 2.0 / AJAX applications such as iTunes and Google Maps will fail in unpredictable ways, especially with schemes that divide the available ports into equally sized port ranges per customer
- Incoming port negotiations may fail – e.g. Universal Plug and Play (UPnP)
- Incoming connections to Well-Known Ports will not work (e.g. SMTP, HTTP, SIP, etc.)
- Reverse DNS pretty much breaks down completely
- Inbound ICMP will fail in most cases
- Security issues are even worse than with CPE NAT
- Packet fragmentation requires special handling
- There are more single points of failure and decreased network stability
- Port randomization is affected (especially in schemes that restrict ports to ranges)
- Penalty Boxes no longer work
- Spam blacklisting will affect many other nodes that use the same address
- Geo-location services may not be reliable or particularly specific
- Load balancing algorithms are impacted
- Authentication mechanisms are impacted
- IPv6 transition mechanisms will be affected (Dual-Stack Lite is the exception here)
- Frequent keep-alives will reduce battery life in mobile nodes

Applications that had to be modified to support NAT Traversal to work through NAT44 will have to be modified *once again*, with even more complicated schemes, to traverse multiple layers of NAT. Application Layer Gateway workarounds now have to be implemented at the Carrier, not just at the Customer Premise. ALGs that have to deal with port-range restrictions will have an even harder job.

Blocking incoming access to services based on IPv4 address will likely affect many "innocent bystanders" that happen to share the same real IPv4 address. One obvious example is spam blacklists. A less obvious example is that some secure devices restrict access by source IP address (only *this* node can connect to my firewall). Now, many other nodes, even in different organizations, will be sharing that same IP address legitimately, so may be able to access such nodes.

With reverse DNS, you publish the node name associated with a given IP address. With CPE NAT this affected many nodes, but this will be completely meaningless for nodes behind Carrier based NAT. There is no way to publish thousands of node names for a single IP address, nor is there any way for someone asking for the reverse lookup to interpret the response correctly.

IPv6 transition mechanisms such as 6to4 will not work at all behind Carrier based NAT, but Teredo might. Likewise IPv4 Multicast and Mobile IPv4 will have to be modified extensively for Carrier based NAT.

Chapter 5 – TCP/IPv6 Core Protocols

This chapter introduces the new concepts and technical specifics of TCP/IPv6, the foundation of the Second Internet. Since IPv6 is based heavily on TCP/IPv4, the approach will be to describe the differences between the two. The subchapter headings are intentionally similar to those in Chapter 3, to allow you to compare the old and the new, topic by topic. Again, there is no intent to be comprehensive. There is a lot of content available on all aspects of IPv6, listed in the bibliography, and/or available online. The ultimate references are the RFCs, so this chapter includes pointers to the relevant ones, for those who want to drill deeper on specific topics.

In other chapters we will discuss topics such as advanced aspects of IPv6 (IPsec, Mobile IPv6), the new things that TCP/IPv6 makes possible, who is involved in making it happen, and how we get from the First Internet to the Second Internet (migration). This chapter covers the core protocols of IPv6.

5.1 – Network Hardware

Essentially the same network hardware that was used to deploy the First Internet is being used to deploy the Second Internet, with some notable exceptions, primarily hardware that implements things at the Internet Layer or above, such as smart ("Layer 3") switches, routers and firewalls. Also DNS and DHCP servers must be updated or replaced with ones that support TCP/IPv6 (more typically both TCP/IPv4 and TCP/IPv6, or "dual stack"). As TCP/IPv6 is deployed, Virtual Private Networks (VPNs) will likely move away from "SSL/VPN" to IPsec based VPNs, which is the only IETF approved technology for VPNs. Unfortunately IPsec is incompatible with NAT, which is now endemic in the First Internet. VoIP and IPTV appliances will probably be upgraded to (or replaced with) TCP/IPv6 based systems. Any device with TCP/IP hardware acceleration (such as in high end routers) will probably need to be redesigned or replaced. Simply upgrading the firmware will not be sufficient on such products. There are some routers that only have hardware acceleration for the IPv4 stack, which has led some people to think there are performance issues with IPv6. Already a number of hardware acceleration chips that support both IPv4 and IPv6 are available and are being used in new product designs.

The hardware of most ***nodes*** does not need to change, especially client and server computers. Replacement or upgrade of the operating system and applications is all that is needed. The good news is that almost all operating systems and many network applications that run on client computers are *already* fully compliant with TCP/IPv6, and those are widely deployed. Those that aren't yet compliant can be upgraded or configured to support it with very reasonable effort and cost. Many server applications (especially open source ones) are already compliant as well. Virtually everything Microsoft makes fully supports TCP/IPv6 today. For client computers, Windows Vista and Windows 7 have very complete support. Windows XP has some support, but is missing some key features (like GUI configuration of IPv6 addresses, and DNS queries over IPv6). My company supplies free tools for Windows XP GUI IPv6 configuration, and a Windows XP DHCPv6 client. Go to *www.infoweapons.com* (downloads) for details. For server computers, Windows Server 2008 and Exchange Server 2007 (and most other server software since 2007) have full support for TCP/IPv6. Most Open Source operating systems (Linux, FreeBSD, OpenBSD and NetBSD) have had full support for TCP/IPv6 for many years. Most

open source network applications (Apache, Nagios, Postfix, Dovecot, etc.) also have full support (although in some cases, documentation may be hard to find).

NICs do not need to change unless they have IPv4 specific hardware acceleration, and even those will typically run TCP/IPv6 with no problem, but the IPv6 part won't be accelerated (it will run at "software" performance levels, in terms of packets or bytes processed per second). There are already many chips available to build hardware accelerated NICs that fully support both TCP/IPv4 *and* TCP/IPv6, so soon, even NICs with hardware acceleration will be no problem. They will accelerate IPv4 and/or IPv6 traffic. For the most part, NICs work at the *Link Layer*, hence are IP version agnostic (except for hardware acceleration).

Existing ***Wi-Fi NICs*** are also IP version agnostic (they work at the *Link Layer*), and every one I've tried has worked with IPv6 with no upgrades or workarounds required. **Wi-Fi Access Points** are another matter, because usually they include higher layer functionality such as IPv4 routing, often including IPv4 NAT and a DHCPv4 server. Even here, there is a simple workaround. Most Wi-Fi access points have a "WAN" connector which is the input to the NAT gateway, and one or more "LAN" connectors that are on the *client* side of the NAT gateway. The LAN connectors are intending to plug in wired client nodes, which are peers to the wireless client nodes (both wired and wireless client nodes obtain configuration information and translated IP addresses from the DHCPv4 server and NAT gateway built into the Wi-Fi access point). Of course the existing IPv4 routing, IPv4 NAT and DHCPv4 in such devices are not compatible with TCP/IPv6. There will be dual stack Wi-Fi Access Points available soon from companies like D-Link, but the majority of products available today do not have routing, firewall or DHCP support for IPv6.

However, if you plug the cable from your ISP DSL modem (or from a larger home wired network) into *one of the LAN connectors* on your Wi-Fi Access Point, instead of into the WAN connector as you are supposed to, you can simply ignore the IPv4 specific parts of the Wi-Fi access point. The actual Wi-Fi transmitter part is IP agnostic, and if there is both TCP/IPv4 and TCP/IPv6 on the feed you connect, they will both be broadcast on wireless, and all existing nodes with Wi-Fi NICs will receive it (assuming each OS supports IPv6 and you have configured it). Of course, if you want your Wi-Fi nodes to obtain IPv4 addresses automatically, you must have a DHCPv4 server somewhere in your network (properly configured). Your Wi-Fi Access Point is no longer performing this function. Likewise if you want Wi-Fi clients to obtain IPv6 addresses through stateless auto configuration, there must be a Router Advertisement Daemon in your network (just as for wired IPv6). If your wireless node has a DHCPv6 client, and you have a DHCPv6 server in your network, stateful auto configuration will work over Wi-Fi as well. Of course you can manually configure IPv6 addresses for Wi-Fi nodes just as you can with wired nodes. No NAT is required for IPv6. For IPv4, no NAT will be performed in the Wi-Fi Access Point, so if you need it, it must be performed at the outside gateway (for example, a wired DSL modem from your ISP). Your wireless nodes will be *peers* to your wired nodes. All of them (wired and wireless) will get addresses from the same DHCPv4 pool (if you use DHCP) and all will be in the same subnet. Normally if you connected a Wi-Fi gateway with NAT inside an existing NATted network, your wireless nodes would be behind *two* levels of NAT, which can cause some problems.

You will also find that some consumer devices that support Wi-Fi already have support for IPv6, such as certain Nokia phones and any phone based on Microsoft Mobile (Samsung Omnia, HTC, etc.). It's kinda cool to deploy dual stack Wi-Fi and show people the dancing turtle at www.kame.net on your phone. With most of today's phones, however, the only thing that works over IPv6 today (if anything) is Wi-Fi

Internet access, not the voice traffic or "Internet over wireless" service. In theory you could add a dual stack softphone (VoIP client) and do voice communications over IPv6, but only via the Wi-Fi connection through a Wi-Fi Access Point connected to the main Internet, not over your wireless telephone carrier's Internet service via WAP, GPRS, EDGE , HSDPA, or whatever else they provide. Someday even these services will be dual stack (probably primarily HSDPA).

Soon there will be dual stack Wi-Fi Access Points that fully support routing for IPv4 and IPv6, NAT for IPv4, and a Router Advertisement Daemon to enable IPv6 stateless auto configuration. D-Link in Taiwan is working on those now, and they should be on the market shortly.

Network cables are totally IP version agnostic. You will not need to rewire you network just for IPv6.

All conventional ("layer 2") ***hubs and switches*** are IP version agnostic, although "layer 3" features of some switches (such as Web management, SNMP, and VLANs) must be upgraded to support IPv6. In most cases, this will be possible simply with new downloaded firmware. No hardware changes are needed (assuming there is sufficient RAM and ROM to handle the more complex firmware). Contact your switch vendor and demand that they add support for IPv6. There are already a few layer 3 switches on the market that support IPv6. I have an SMC 8848M 48-port Gigabit managed switch in my home network that has quite a bit of IPv6 support, including web management over IPv6, IPv6 based VLANs, SNMP over IPv6, etc. Unfortunately, traffic statistics do not breakout IPv4 and IPv6 traffic, just the total is reported. D-Link also recently announced a dual stack smart switch series. They are already *IPv6 Ready Gold* certified. For details, search for their DGS-3627 XSTACK Managed 24 port Gigabit Stackable L3 Switch.

Many ***enterprise grade routers*** and ***firewalls*** already support TCP/IPv6, although in some cases you must pay extra for the IPv6 functionality. Cisco routers require "Advanced IP Services" for IOS, usually at additional cost, before IPv6 works. For example, the Cisco 2851 router ($6495) includes only the base IOS (no IPv6 support). The Advanced IP Services Feature Pack for it is an additional $1700 (all prices list). When buying or considering using Cisco Routers for use in IPv6 networks, make sure they already include Advanced IP Services, or include the additional cost of the Feature Pack.

Home network gateways that support TCP/IPv6 are further behind, but coming soon, especially from Asian vendors, such as D-Link. A typical one will have all the features of existing IPv4 based gateways, plus 6in4 tunneling (to tunnel in IPv6 from a virtual ISP), a Router Advertisement Daemon (to enable stateless auto configuration), and firewall rules for IPv6 traffic. They should also be able to accept direct (as opposed to tunneled) IPv6 service, for when dual stack ISP service becomes more widely available. Their DNS relay should support DNS over both IPv4 and IPv6. More advanced gateways might include a DHCPv6 server.

Note that some DSL or cable modems also include IPv4 firewall functionality. Of course this will not allow you to control IPv6 traffic. Therefore, if you are connecting your LAN to the IPv6 Internet, there must be IPv6 firewalling somewhere, possibly in a 6in4 tunnel endpoint that is routing IPv6 traffic into your LAN. A Dual Stack gateway firewall may include routing to accept incoming "direct" IPv6 service and/or a 6in4 endpoint to accept incoming "tunneled" IPv6 service, together with both IPv4 and IPv6 filtering rules, and a Router Advertisement Daemon to support stateless auto configuration for the internal nodes that support IPv6. My company makes an easy to use dual stack firewall with all of these

features, and an intuitive GUI administrative interface (SolidWall). There are several such products on the market today, in addition to open source projects you can install on Linux or FreeBSD.

You can find out more about D-Link's IPv6 compliant products at:

http://www.dlink.com/business/IPv6/

Most ***IP Phones*** in use today do not support TCP/IPv6, but some new models (including ones from Snom in Germany and Moimstone in Korea) do support it. Cisco supports IPv6 on a number of their recent phones, including the 7906G, 7911G, 7931G, 7941G/GE, 7942G, 7945G, 7961G/GE, 7962G, 7965G, 79770G, 7971G/GE, and 7975G. Most of the older Cisco IP phones currently in use do not support IPv6, and their firmware cannot be upgraded for various reasons.

When looking for hardware products that already support TCP/IPv6, an excellent source of information is the *IPv6 Ready Approved Products List*. If possible, choose products that have passed the Phase 2 (Gold level) testing. This insures full compliance with all relevant RFCs and interoperability with many other products. There is also a list of products that have passed the Phase 1 (Silver level) testing. Phase 1 testing insures compliance with all items denoted MUST in the relevant RFCs. Phase 2 testing *also* insures compliance with all items denoted SHOULD in the relevant RFCs (a much more comprehensive set of functionality). These lists are updated and maintained by the IPv6 Ready Logo Committee of the IPv6 Forum. They can be found here:

http://www.IPv6ready.org/phase-1_approved_list
http://www.IPv6ready.org/phase-2_approved_list

5.2 – RFCs: A Whole Raft of New Standards for TCP/IPv6

There are many new RFCs that define the protocols, addressing and routing schemes, as well as migration issues for TCP/IPv6. I will cover the most important of those in this chapter.

You can trace the beginnings and evolution of TCP/IPv6 in some early RFCs. In 1990, when the IETF first realized that a successor to TCP/IPv4 was going to be needed (and soon), the fun began. One key RFC related to this is RFC 1752, "The Recommendation for the IP Next Generation Protocol", January 1995. Prior to this, people referred to the successor protocol as IPng (IP next generation), but in this RFC the term IPv6 was used. RFC 1752 says that the IETF started its effort to select a successor in late 1990, and that several parallel efforts were started. Among these proposals were "CNAT", "IP Encaps", "Nimrod", "Simple CLNP", the "P Internet Protocol", the "Simple Internet Protocol" and "TP/IX". None of these ever made it past the Internet Draft stage.

By late 1993, an IPng Working Group was formed, and the various proposals still around were reviewed. These included CATNIP, TUBA, and SIPP. Relevant RFCs (now of only historical interest) are:

- RFC 1347 "TCP and UDP with Bigger Addresses (TUBA)", June 1992 (Informational)
- RFC 1526 "Assignment of System Identifiers for TUBA/CLNP Hosts", September 1993 (Informational)
- RFC 1561, "Use of ISO CLNP in TUBA Environments", December 1993 (Experimental)

- RFC 1707, “CATNIP: Common Architecture for the Internet”, October 1994 (Informational)
- RFC 1710, “Simple Internet Protocol Plus White Paper”, October 1994 (Informational)

The CLNP referred to in several of these was the “Connectionless-mode Network Layer Protocol”, defined in ISO/IEC 8473, which did not make it into the final IPv6 specification. By 1995 a consensus had emerged, with the best features of all the contenders. The consensus was summarized in RFC 1752. Before the end of the year (barely), the first real TCP/IPv6 specifications were published:

- RFC 1883, “Internet Protocol, Version 6 (IPv6) Specification”, December 1995 (Standards Track, obsoleted by RFC 2460)
- RFC 1884, “IP Version 6 Addressing Architecture”, December 1995 (Standards Track, obsoleted by RFC 2373)
- RFC 1885, “Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification”, December 1995 (Standards Track, obsoleted by RFC 2463)
- RFC 1886, “DNS extensions to support IP version 6”, December 1995 (Standards Track – obsoleted by RFC 3596)
- RFC 1887, “An Architecture for IPv6 Unicast Address Allocation”, December 1995 (Informational)

Most of these have been updated since then and there are quite a few new ones since 1995, but this is where it really started. Yes, TCP/IPv6 is turning 15 years old in 2010, and has finally grown up.

5.3 – TCP/IPv6

The software that is making the Second Internet (and virtually all Local Area Networks) possible will be around for quite some time. Like its predecessor, TCP/IPv4, it is a suite (family) of protocols. Once again, the core protocols are TCPv6 (Transmission Control Protocol version 6) and IPv6 (Internet Protocol version 6). TCPv6 has very few changes from TCPv4, but there are a few, due to the larger addresses that require more storage, and the odd method of calculating the checksum defined in TCPv4 (this involves a “pseudo header” that includes the source and destination addresses from the IP header, which of course are different in IPv4 and IPv6).

There is no new RFC specifically about TCPv6, but there are several RFCs that include details about the new features.

UDP has only very minor changes to work over IPv6, primarily to provide more storage for IPv6 addresses. The UDP packet header checksum also includes the IP addresses, once again using the new pseudo header.

The following standards current define IPv6:

- RFC 1809, “Using the Flow Label Field in IPv6”, June 1995 (Informational)
- **RFC 1881, “IPv6 Address Allocation Management”, December 1995 (Informational)**
- **RFC 1887, “An Architecture for IPv6 Unicast Address Allocation”, December 1995 (Informational)**
- RFC 1981, “Path MTU Discovery for IP version 6”, August 1996 (Standards Track)

- RFC 2428, “FTP Extensions for IPv6 and NATs”, September 1998 (Standards Track)
- **RFC 2460, “Internet Protocol, Version 6 (IPv6) Specification”, December 1998 (Standards Track)**
- RFC 2473, “Generic Packet Tunneling in IPv6 Specification”, December 1998 (Standards Track)
- **RFC 2474, “Definition of the Differentiated Service Field (DS Field) in the IPv4 and IPv6 Headers”, December 1998 (Standards Track)**
- RFC 2526, “Reserved IPv6 Subnet Anycast Addresses”, March 1999 (Standards Track)
- RFC 2529, “Transmission of IPv6 over IPv4 Domains without Explicit Tunnels”, March 1999 (Standards Track)
- RFC 2675, “IPv6 Jumbograms”, August 1999 (Standards Track)
- RFC 2711, “IPv6 Router Alert Option”, October 1999 (Standards Track)
- RFC 2765, “Stateless IP/ICMP Translation Algorithm (SIIT)”, February 2000 (Standards Track)
- **RFC 2767, “Dual Stack Hosts using the Bump-In-the-Stack Technique (BIS)”, February 2000 (Informational)**
- RFC 2894, “Router Renumbering for IPv6”, August 2000 (Standards Track)
- **RFC 3053, “IPv6 Tunnel Broker”, January 2001 (Informational)**
- RFC 3056, “Connection of IPv6 Domains via IPv4 Clouds”, February 2001 (Standards Track)
- RFC 3089, “A SOCKS-based IPv6/IPv4 Gateway Mechanism”, April 2001 (Informational)
- RFC 3111, “Service Location Protocol Modifications for IPv6”, May 2001 (Standards Track)
- **RFC 3122, “Extensions to IPv6 Neighbor Discovery for Inverse Discovery Specification”, June 2001 (Standards Track)**
- RFC 3142, “An IPv6-to-IPv4 Transport Relay Translator”, June 2001 (Informational)
- RFC 3175, “Aggregation of RSVP for IPv4 and IPv6 Reservations”, September 2001 (Standards Track)
- RFC 3177, “IAB/IESG Recommendations on IPv6 Address Allocations to Sites”, September 2001 (Informational)
- RFC 3178, “IPv6 Multihoming Support at Site Exit Routers”, October 2001 (Informational)
- **RFC 3306, “Unicast-Prefix-based IPv6 Multicast Addresses”, August 2002 (Standards Track)**
- RFC 3314, “Recommendations for IPv6 in Third Generation Partnership Project (3GPP) Standards”, September 2002 (Informational)
- RFC 3316, “Internet Protocol Version 6 (IPv6) for Some Second and Third Generation Cellular Hosts”, April 2003 (Informational)
- **RFC 3363, “Representing Internet Protocol version 6 (IPv6) Addresses in the Domain Name System”, August 2002 (Informational)**
- RFC 3364, “Tradeoffs in Domain Name System (DNS) Support for Internet Protocol version 6 (IPv6)”, August 2002 (Informational)
- RFC 3484, “Default Address Selection for Internet Protocol version 6 (IPv6)”, February 2003 (Standards Track)
- RFC 3531, “A Flexible Method for Managing the Assignment of Bits of an IPv6 Address Block”, April 2003 (Informational)
- RFC 3574, “Transition Scenarios for 3GPP Networks”, August 2003 (Informational)
- RFC 3582, “Goals for IPv6 Site-Multihoming Architectures”, August 2003 (Informational)
- **RFC 3587, “IPv6 Global Unicast Address Format”, August 2003 (Informational)**
- RFC 3595, “Textual Conventions for the IPv6 Flow Label”, September 2003 (Standards Track)
- **RFC 3697, “IPv6 Flow Label Specification”, March 2004 (Standards Track)**

- RFC 3710, “6bone (IPv6 Testing Address Allocation) Phaseout”, March 2004 (Standards Track)
- RFC 3750, “Unmanaged Networks IPv6 Transition Scenarios”, April 2004 (Informational)
- RFC 3756, “IPv6 Neighbor Discovery (ND) Trust Models and Threats”, May 2004 (Informational)
- RFC 3769, “Requirements for IPv6 Prefix Delegation”, June 2004 (Informational)
- RFC 3849, “IPv6 Address Prefix Reserved for Documentation”, July 2004 (Informational)
- RFC 3879, “Deprecating Site Local Addresses”, September 2004 (Standards Track)
- RFC 3904, “Evaluation of IPv6 Transition Mechanisms for Unmanaged Networks”, September 2004 (Informational)
- RFC 3974, “SMTP Operational Experience in Mixed IPv4/v6 Environments”, January 2005 (Informational)
- **RFC 4007, “IPv6 Scoped Address Architecture”, March 2005 (Informational)**
- RFC 4029, “Scenarios and Analysis for Introducing IPv6 into ISP Networks”, March 2005 (Informational)
- RFC 4038, “Application Aspects of IPv6 Transition”, March 2005 (Informational)
- RFC 4057, “IPv6 Enterprise Network Scenarios”, June 2005 (Informational)
- RFC 4074, “Common Misbehavior Against DNS Queries for IPv6 Addresses”, May 2005 (Informational)
- RFC 4135, “Goals of Detecting Network Attachment in IPv6”, May 2005 (Informational)
- RFC 4147, “Proposed Changes to the Format of the IANA IPv6 Registry”, August 2005 (Informational)
- RFC 4159, “Depreciation of ip6.in”, August 2005 (Best Current Practice)
- RFC 4177, “Architectural Approaches to Multihoming for IPv6”, September 2005 (Informational)
- RFC 4192, “Procedures for Renumbering an IPv6 Network without a Flag Day”, September 2005 (Informational)
- **RFC 4193, “Unique Local IPv6 Unicast Addresses”, October 2005 (Standards Track)**
- **RFC 4213, “Basic Transition Mechanisms for IPv6 Hosts and Routers”, October 2005 (Standards Track)**
- RFC 4215, “Analysis of IPv6 Transition in Third Generation Partnership Project (3GPP) Networks”, October 2005 (Informational)
- RFC 4218, “Threats Relating to IPv6 Multihoming Solutions”, October 2005 (Informational)
- RFC 4241, “A Model of IPv6/IPv4 Dual Stack Internet Access Service”, December 2005 (Informational)
- **RFC 4291, “IP Version 6 Addressing Architecture”, February 2006**
- RFC 4294, “IPv6 Node Requirements”, April 2006 (Informational)
- RFC 4311, “IPv6 Host-to-Router Load Sharing”, November 2005 (Standards Track)
- RFC 4330, “Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI”, January 2006 (Informational)
- RFC 4339, “IPv6 Host Configuration of DNS Server Information Approaches”, February 2006 (Informational)
- RFC 4380, “Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs)”, February 2006 (Standards Track)
- RFC 4429, “Optimistic Duplicate Address Detection (DAD) for IPv6”, April 2006 (Standards Track)
- **RFC 4443, “Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification”, April 2006 (Standards Track)**
- RFC 4472, “Operational Considerations and Issues with IPv6 DNS”, April 2006 (Informational)

- RFC 4554, “Use of VLANs for IPv4-IPv6 Coexistence in Enterprise Networks”, June 2006 (Informational)
- RFC 4659, “BGP-MPLS IP Virtual Private Network (VPN) Extensions for IPv6 VPN”, September 2006 (Standards Track)
- RFC 4692, “Considerations on the IPv6 Host Density Metric”, October 2006 (Informational)
- RFC 4727, “Experimental Values in IPv4, IPv6, ICMPv4, ICMPv6, UDP and TCP Headers”, November 2006 (Standards Track)
- RFC 4773, “Administration of the IANA Special Purpose IPv6 Address Block”, December 2006 (Informational)
- RFC 4779, “ISP IPv6 Deployment Scenarios in Broadband Access Networks”, January 2007 (Informational)
- RFC 4798, “Connecting IPv6 Islands over IPv4 MPLS Using IPv6 Provider Edge Routers (6PE)”, February 2007 (Standards Track)
- RFC 4818, “RADIUS Delegated-IPv6-Prefix Attribute”, April 2007 (Standards Track)
- RFC 4852, “IPv6 Enterprise Network Analysis – IP Layer 3 Focus”, April 2007 (Informational)
- **RFC 4861, “Neighbor Discovery for IP version 6 (IPv6)”, September 2007 (Standards Track)**
- **RFC 4862, “IPv6 Stateless Address Autoconfiguration”, September 2007 (Standards Track)**
- RFC 4864, “Local Network Protection for IPv6”, May 2007 (Informational)
- RFC 4890, “Recommendations for Filtering ICMPv6 Messages in Firewalls”, May 2007 (Informational)
- RFC 4919, “IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement and Goals”, August 2007 (Informational)
- **RFC 4941, “Privacy Extensions for Stateless Address Autoconfiguration in IPv6”, September 2007 (Standards Track)**
- RFC 4942, “IPv6 Transition/Co-existence Security Considerations”, September 2007 (Informational)
- RFC 4943, “IPv6 Neighbor Discovery On-link Assumption Considered Harmful”, September 2007 (Informational)
- RFC 4968, “Analysis of IPv6 Link Models for 802.16 Based Networks”, August 2007 (Informational)
- RFC 5006, “IPv6 Router Advertisement Option for DNS Configuration”, September 2007 (Experimental)
- RFC 5095, “Deprecation of Type 0 Routing Headers in IPv6”, December 2007 (Standards Track)
- RFC 5156, “Special-Use IPv6 Addresses”, April 2008 (Informational)
- RFC 5157, “IPv6 Implications for Network Scanning”, March 2008 (Informational)
- RFC 5172, “Negotiation for IPv6 Datagram Compression Using IPv6 Control Protocol”, March 2008 (Standards Track)
- RFC 5175, “IPv6 Router Advertisement Flags Option”, March 2008 (Standards Track)
- RFC 5181, “IPv6 Deployment Scenarios in 802.16 Networks”, May 2008 (Informational)
- **RFC 5214, “Intra-Site Automatic Tunnel Addressing Protocol (ISATAP)”, March 2008 (Informational)**
- RFC 5350, “IANA Considerations for the IPv4 and IPv6 Router Alert Options”, September 2008 (Standards Track)
- RFC 5375, “IPv6 Unicast Address Assignment Considerations”, December 2008 (Informational)
- RFC 5453, “Reserved IPv6 Interface Identifiers”, February 2009 (Standards Track)

- RFC 5533, “Shim6: Level 3 Multihoming Shim Protocol for IPv6”, June 2009 (Standards Track)
- RFC 5534, “Failure Detection and Locator Pair Exploration Protocol for IPv6 Multihoming”, June 2009 (Standards Track)
- RFC 5549, “Advertising IPv4 Network Layer Reachability Information with an IPv6 Next Hop”, May 2009 (Standards Track)
- **RFC 5569, “IPv6 Rapid Deployment on IPv4 Infrastructures (6rd)”, January 2010 (Informational)**
- RFC 5570, “Common Architecture Label IPv6 Security Option (CALIPSO)”, July 2009 (Informational)
- **RFC 5572, “IPv6 Tunnel Broker with the Tunnel Setup Protocol (TSP)”, February 2010 (Experimental)**
- **RFC 5579, “Transmission of IPv4 Packets over Intra-Site Automatic Tunnel Addressing Protocol (ISATAP) Interfaces”, February 2010 (Informational)**
- RFC 5619, “Softwire Security Analysis and Requirements”, August 2009 (Standards Track)
- RFC 5701, “IP Address Specific BGP Extended Community Attribute”, November 2009 (Standards Track)
- RFC 5722, “Handling of Overlapping IPv6 Fragments”, December 2009 (Standards Track)
- RFC 5798, “Virtual Router Redundancy Protocol (VRRP) Version 3 for IPv4 and IPv6”, March 2010 (Standards Track)

5.3.1 – Four Layer TCP/IPv6 Architectural Model

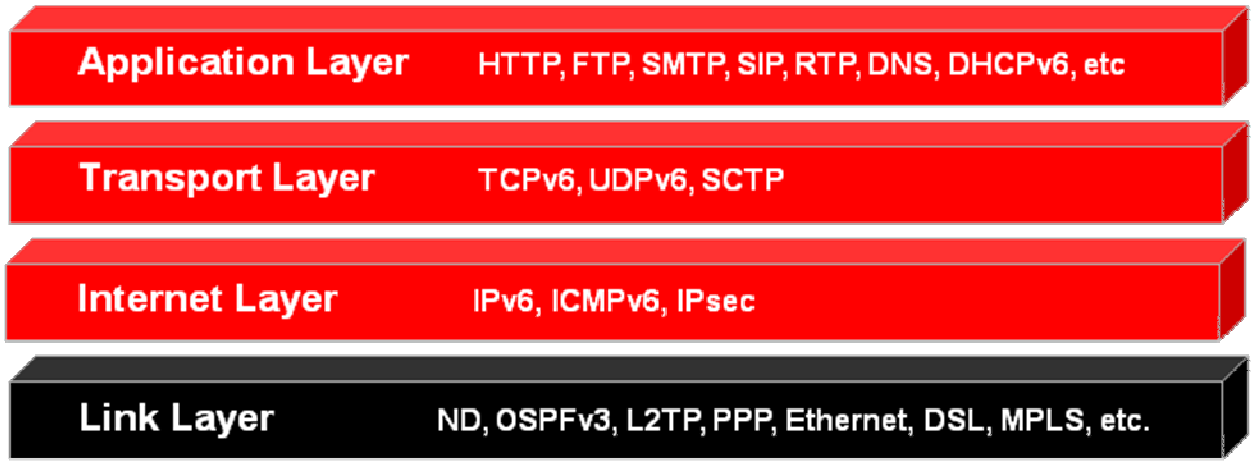


Figure 5.3-a: Four Layer TCP/IPv6 Model

The major changes from the TCP/IPv4 model are:

- Application Layer: DHCPv4 replaced with DHCPv6
- Transport Layer: TCPv4 replaced with TCPv6, UDPv4 replaced with UDPv6
- Internet Layer: IPv4 replaced with IPv6, ICMPv4 replaced with ICMPv6
- Link Layer: Removed ARP, Added ND, OSPFv2 replaced with OSPFv3

In the following discussion, traffic really flows both *down* from the Application Layer to the Link Layer (then out the wire), *and* from the wire up through the Link Layer to the Application Layer. For clarity, only the downward path is described below. When traffic goes up through the layers, each layer strips off one header and hands off the remaining bytes to the layer above.

The *Application Layer* implements the protocols most people are familiar with (e.g. HTTP). The software routines for these are typically contained in application programs such as browsers or web servers that make system calls to subroutines (or "functions" in C terminology) in the "socket API" (an API is an Application Program Interface, or a collection of related subroutines, typically supplied with the operating system or programming language). The application code creates outgoing data streams, and then calls routines in the API to actually send the data via TCP (Transmission Control Protocol) or UDP (User Datagram Protocol). Output to *Transport Layer*: *[DATA]* using IP addresses.

The *Transport Layer* implements TCP (the Transmission Control Protocol) and UDP (the User Datagram Protocol). These routines are internal to the Socket API. They add a TCP or UDP packet header to the data passed down from the *Application Layer*, and then pass the data down to the *Internet Layer* for further processing. Output to *Internet Layer*: *[TCP HDR [DATA]]*, using IP addresses.

The *Internet Layer* implements IPv6 (the Internet Protocol) and various other related protocols such as ICMPv6 (which includes the "ping" function among other things). The IP routine takes the data passed down from the *Transport Layer* routines, adds an IPv6 packet header onto it, then passes the now complete IPv6 packet down to routines in the *Link Layer*. Output to Link layer: *[IPv6 HDR [TCP HDR [DATA]]]* using IP addresses.

The *Link Layer* implements ND (the Neighbor Discovery protocol) that help locates the link layer addresses of other nodes on the link, in addition to other functionality. It also contains routines that actually read and write packets (as fed down to it by routines in the *Internet Layer*) onto the network wire, in compliance with Ethernet or other standards. Output to wire: Ethernet frame using MAC addresses (or the equivalent if other network hardware is used, such as Wi-Fi), which includes the entire IPv6 packet.

The following standards are relevant to the Link Layer in IPv6:

- **RFC 2464, "Transmission of IPv6 Packets over Ethernet Networks", December 1998 (Standards Track)**
- RFC 2467, "Transmission of IPv6 Packets over FDDI Networks", December 1998 (Standards Track)
- RFC 2470, "Transmission of IPv6 Packets over Token Ring Networks", December 1998 (Standards Track)
- RFC 2491, "IPv6 over Non-Broadcast Multiple Access (NBMA) networks", January 1999 (Standards Track)
- RFC 2492, "IPv6 over ATM Networks", January 1999 (Standards Track)
- RFC 2497, "Transmission of IPv6 Packets over ARCnet Networks", January 1999 (Standards Track)
- RFC 2590, "Transmission of IPv6 Packets over Frame Relay Networks Specification", May 1999 (Standards Track)

- RFC 3146, “Transmission of IPv6 Packets over IEEE 1394 Networks”, October 2001 (Standards Track)
- RFC 4338, “Transmission of IPv6, IPv4 and Address Resolution Protocol (ARP) Packets over Fibre Channel”, January 2006 (Standards Track)
- RFC 4392, “IP over InfiniBand (IPoIB) Architecture”, April 2006 (Informational)
- RFC 4944, “Transmission of IPv6 Packets over IEEE 802.15.4 Networks”, September 2007 (Standards Track)
- **RFC 5072, “IP Version 6 over PPP”, September 2007 (Standards Track)**
- RFC 5121, “Transmission of IPv6 via the IPv6 Convergence Sublayer over IEEE 802.16 Networks”, February 2008 (Standards Track)

5.3.2 – IPv6: The Internet Protocol, Version 6

IPv6 is the foundation of TCP/IPv6 and accounts for many of its distinguishing characteristics, such as its 128-bit address size, its addressing model, its packet header structure and routing. IPv6 is currently defined in RFC 2460, “Internet Protocol, Version 6 (IPv6) Specification”, December 1998, but there are several RFCs that extend the definition.

5.3.2.1 – IPv6 Packet Header Structure

So what are these packet headers mentioned above? In TCP/IPv6 packets, there is an IPv6 packet header, then zero or more packet header extensions, then a TCP or UDP header, and finally the packet data. Each header and header extension is a structured collection of data, including things such as the IPv6 address of the sending node, and the IPv6 address of the destination node. Why are we getting down to this level of detail? Because some of the big changes from IPv4 to IPv6 have to do with the new and improved IP packet header architecture in IPv6. In this chapter, we’ll cover the IPv6 packet header. Here it is:

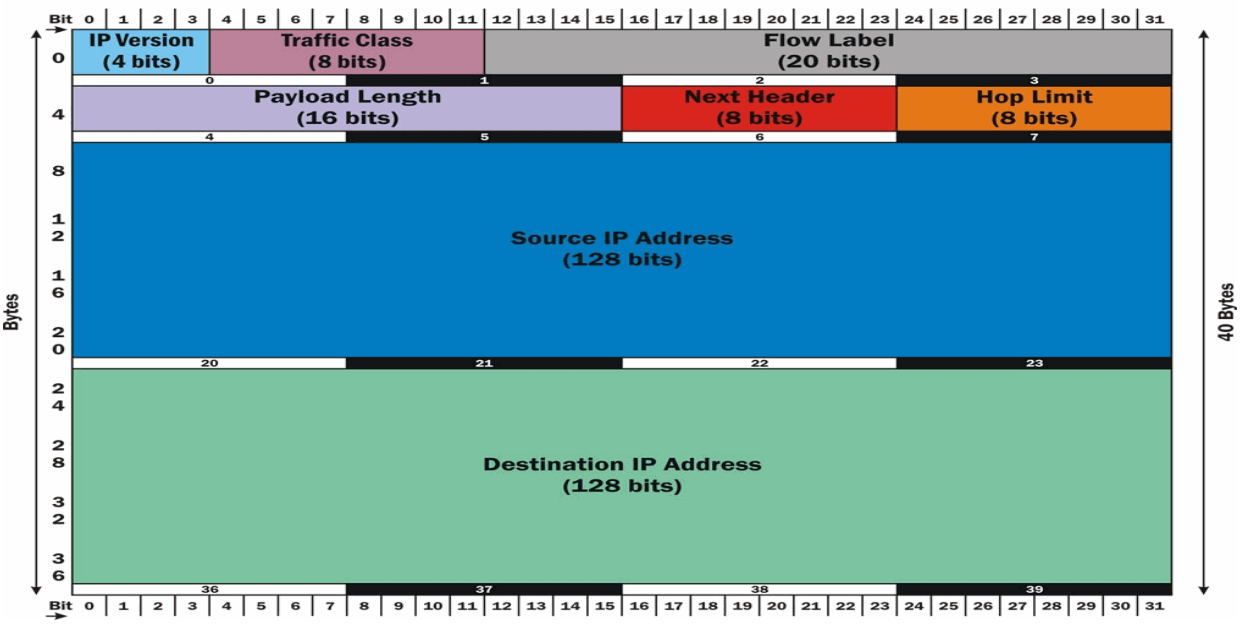


Figure 5.3-b: IPv6 Packet Header

The *IP Version* field (4 bits) contains the value 6 (imagine that!) which in binary is "0110". This field allows IPv4 and IPv6 traffic to be mixed in a single network.

The *Traffic Class* field (8 bits) Available for use by originating nodes and/or forwarding routers to identify and distinguish between different classes or priorities of IPv6 packets, in a manner virtually identical to that of IPv4 "Type of Service".

The *Flow Label* field (20 bits) is something new in IPv6. It can be used to tag up to 2^{20} (1,048,576) distinct traffic flows, for purposes such as fine grained bandwidth management (QoS). Its use is still experimental. Hosts or routers that do not support this function should set it to zero when originating a packet, or ignore it when receiving a packet. The semantics and usage of this field are covered in Appendix A of RFC 2460.

The *Payload Length* field (16 bits) is the length of the IPv6 packet payload in bytes, not counting the standard packet header (as it is in IPv4 *Total Length*), but *counting* the size of any extension headers, which don't exist in IPv4. You can think of packet extension headers as being the first part of the data field (payload) of the IPv6 packet.

The *Next Header* field (8 bits) indicates the type of header immediately following the standard IPv6 packet header. It uses the same values as the IPv4 *Protocol* field, as defined in RFC 1700, "Assigned Numbers", October 1994. If this value contains the code for *TCP*, then the TCP header and packet payload (data) begins immediately after the IPv6 packet header. Otherwise one or more IPv6 extension headers will be found before the TCP header and data begins. Since each extension header has another *Next Header* field (and a *Header Length* field), this constitutes a linked list of headers before the final extension header, which is followed by the data. UDP packets can also have extension headers.

The *Hop Limit* field (8 bits) is to prevent packets from being shuttled around indefinitely on a network. Every time a packet crosses a switch or router, the hop count is decremented by one. If it reaches zero, the packet is dropped. Typically if this happens, an ICMPv6 message ("time exceeded") is returned to the packet sender. This mechanism is how the *traceroute* command works.

The *Source Address* field (128 bits) contains the IPv6 address of the packet sender.

The *Destination Address* field (128 bits) contains the IPv6 address of the packet recipient.

Note: the following fields from the IPv4 packet header have been eliminated in the IPv6 packet header: *Header Length*, *Identification (Fragment ID)*, *Fragmentation Flags*, *Fragment Offset*, *Header Checksum*, and *Options*. The value in the *Payload Length* field no longer includes the length of the standard packet header. The *Flow Label* field had no corresponding field in the IPv4 packet header. Some of the missing fields (e.g. fragmentation information) have been pushed into extension packet headers. For example, in IPv6 only fragmented packets have the fragmentation header extension. Unfragmented packets do not have to carry the unnecessary overhead. In IPv4, all packets have the fragmentation fields in their header, whether they are fragmented or not.

IPv6 Packet Fragmentation and Path MTU Discovery

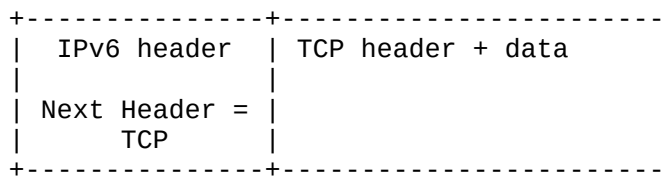
The fields related to fragmentation are now found in the fragmentation extension header, which exists only in fragmented packets (no need to clutter up unfragmented packets, as in IPv4). In IPv6, only the originating node can fragment packets (no intervening node is supposed to do this). The originating node uses *MTU Path Discovery* to determine the "width" of the proposed path (the maximum packet size that it can handle). *MTU* stands for Maximum Transmitted Unit (maximum packet length). Any packets larger than that size must be fragmented before transmission by the originating node, and reassembled upon receipt by the destination node. There is a default packet size that any IPv6 node must be able to handle (1280 bytes). MTU Path Discovery allows the sender to determine if larger (more efficient) packets can be used. The originating node assumes the Path MTU is the MTU of the first hop in the path. A trial packet of this size is sent out. If any link is unable to handle it, an ICMPv6 *Packet Too Big* message is returned. The originating node iteratively tries smaller packet sizes until it gets no complaints from any node, and then uses the largest MTU that was acceptable along the entire path. This process takes place automatically in the *Internet Layer*. There is no corresponding mechanism in IPv4.

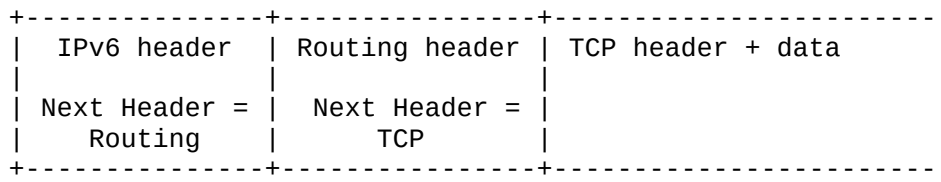
Extension Headers (new in IPv6)

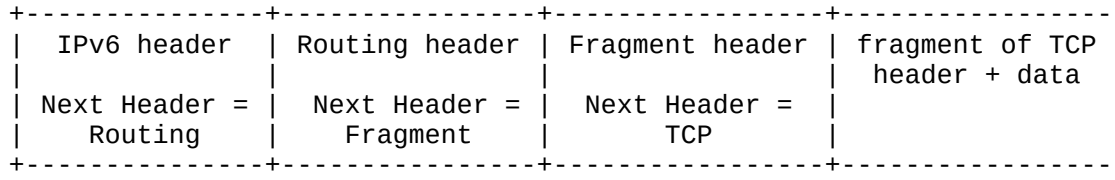
After the main header, there can be zero or more *extension headers*, before the payload (actual packet data). This approach makes IPv6 highly extensible, for new functionality in years to come. Several extension headers are already defined, and doubtless more will be defined over time.

The first byte of each extension header contains a *Next Header* field, identical to the same named field in the main IPv6 packet header (using codes from RFC 1700). The second byte of each extension header contains a *Header Extension Length* field, which specifies the length of this header, in 8 byte units, not including the first 8 bytes. Thus every extension header is at least 8 bytes long, and is a multiple of 8 bytes in length. The following header (or data, if no more extension headers) will begin immediately after the end of this extension header. This effectively defines a *linked list* (a data structure familiar to all programmers).

Here are some typical packet header sequences to illustrate how each chains to the next:







The basic Extension Headers are defined in RFC 2460, “Internet Protocol, Version 6 (IPv6) Specification”, December 1998. These include the following:

- Options Extension Header
- Hop-by-hop Options Extension Header
- Routing Extension Header
- Fragment Extension Header
- Destination Extension Header

Two extension headers are used for IPsec (IP layer security). The IPsec Authentication extension header (IPsec AH) is defined in RFC 2402, “IP Authentication Header”, November 1998. The Encapsulating Security Payload header (IPsec ESP) is defined in RFC 2406, “IP Encapsulating Security Payload (ESP)”, November 1998.

When multiple extension headers are used in a single packet, the following order should be followed:

- IPv6 basic header
- Hop-by-Hop Options header
- Destination Options header (for options to be processed by more than just final recipient)
- Routing header
- Fragment header
- Authentication header
- Encapsulating Security Payload header
- Destination Options header (for options to be processed only by final recipient)

- Upper Layer header (TCP, UDP or SCTP)

Hop-by-hop Options Header – used to carry optional information that must be examined by every node along a packet's delivery path. This option is indicated by a *Next Header* value of 0.

Routing Header – used by an IPv6 source node to list one or more intermediate nodes to be "visited on the way" to a packet's destination. This is similar to IPv4's Loose Source and Record Route option. The *Routing Header* is identified by a *Next Header* value of 43.

Fragment Header – used by an IPv6 source to send a packet larger than would fit in the path MTU to its destination. In IPv6, packet fragmentation is performed only by the source node, which must use MTU discovery to determine the maximum packet size along the proposed path. The *Fragment Header* is identified by a *Next Header* value of 44.

Destination Options Header – used to carry optional information that needs to be examined only by a packet's destination node(s). The *Destination Options Header* is identified by a *Next Header* value of 60.

For the specific details on each of the above header extension packets, see RFC 2460. The Authentication Header and ESP packet headers will be described later, under IPsec.

5.3.2.2 – IPv6 Addressing Model

In IPv6, addresses are 128 bits in length. They are simply numbers from 0 to about 340 undecillion (340 trillion, trillion, trillion). In exponential notation, that would be 3.40 e+38 (think of it as a 38 digit phone number, where IPv4 is a 9 digit phone number). However you write it, that's a *really* big number. For the convenience of humans, these numbers are typically represented in what I call *coloned hex* notation (as opposed to the *dotted decimal* notation used with IPv4). This splits the 128 bit addresses into eight 16-bit fields, and then represents each field with a hexadecimal (base 16) number from 0 to ffff (you can use upper or lower case for the hexadecimal digits A-F, but it is common practice in IPv6 to use lower case). These hexadecimal numbers cover all possible 16 bit binary patterns from 0000 0000 0000 0000 to 1111 1111 1111 1111. The hexadecimal numbers are separated by colons (":"). Leading zeros can be eliminated in each field. At most one run of zeros can be replaced by the double colon, "::". The following are all valid IPv6 addresses written in coloned hex notation:

```
2001:df8:5403:3000:b5ea:976d:679f:30f5  A unicast address using EUI-64
2001:df8:5403:3000::1e                 A manually assigned unicast address
fe80::b5ea:976d:679f:30f5              A link-local address using EUI-64
ff02::1                                A multicast address (all nodes on link)
::1                                    The loopback address for IPv6
::                                     The unspecified address (all zeros)
```

Some people are aware that you can use IPv4 addresses instead of nodenames in web URIs, for example: http://123.45.67.89/main.html. You can also use IPv6 addresses, but because colons demark other things in URIs (such as non-standard port number), you cannot use IPv6 addresses "as is"; enclose them in square brackets ([]). For example, http://[2001:df8:5403:3000::d]/nagios is a valid URI that includes an IPv6 numeric address.

In certain cases, the size of the subnet is specified after the address, similar to CIDR. This is especially common when representing prefixes, for example:

```
2001:df8:5403::/48                  An organization's 48 bit network prefix
2001:df8:5403:3000::/52             A /52 block routed into a branch office
2001:df8:5403:3000::/64             The 64 bit prefix for one subnet in a branch
```

When an RIR (e.g. APNIC) allocates a "/32" block of addresses to an ISP they assign the first 32 bits of those addresses, based on the next available "/32" block from the unallocated pool at that time. A "/32" block contains 65,536 "/48" blocks to allocate to customers. If the ISP allocates all of those, then the RIR will give them a new "/32" block, each address of which will have a completely different first 32 bits from the addresses in the previous "/32" block given to the ISP. The leftmost, or *most significant* 32 bits of every address in a given "/32" block will all be the same. All addresses from smaller blocks (like a "/48" block or "/64" block) carved out of that "/32" block by the ISP (for allocation to customers) will have the same first 32 bits. For example, many of NTT American's IPv6 allocations include addresses that start with "*2001:418::/32*", including our "/48" block. No other ISP in the world will *ever* be allocated a "/32" block with those particular first 32 bits. Up to 65,536 of NTT America's customers might get "/48" blocks whose addresses start with those 32 bits.

When an ISP allocates a "/48" block for a customer from their "/32" block, the *next* 16 bits (following the first 32) are chosen by that ISP, so that the first 48 bits will be unique to that customer in the entire world. The first 48 bits of every address in a "/48" block given to a end-user organization will all be the same, but will be different from the first 48 bits of the addresses in any other "/48" block in the world. You can think of this 48-bit sequence as the *organization prefix*. All addresses in our "/48" block from NTT America happens to start with "*2001:418:5403::/48*". No other customer of NTT America has 5403 in the third 16 bit field of their addresses. When a customer deploys subnets, they choose a *further* 16 bit value (unique within their organization) for each subnet, which together with the organization's 48 bit prefix, creates a globally unique 64 bit prefix for a working subnet. This can be used to manually configure 128-bit addresses for nodes on that subnet, or can be configured on the Router Advertisement Daemon that supplies prefixes to nodes in that subnet for Stateless Address Autoconfiguration. If using stateful DHCPv6, the administrator can also create pools of addresses for assignment, where each 128-bit address in a pool has that same 64 bit subnet prefix.

IPv6 Packet Transmission Types

In TCP/IPv4, there were several packet transmission types (unicast, anycast and multicast). IPv4 Multicast uses Class D addresses, while all other addresses are unicast (or reserved). There is no real concept of *scope* in IPv4 (the part of the network in which a given address is valid and unique). IPv4 "Private Addresses" are a step in this direction, but IPv6 defines real scope rules for certain kinds of addresses. These concepts are defined in RFC 4291, "IP Version 6 Addressing Architecture", February 2006. Note: in Windows, "ping" is used for both IPv4 and IPv6. In Linux and BSD, the "ping" command is used just for IPv4 – in IPv6, the command is "ping6". In the following, I use just the generic "ping", but be aware that for IPv6 on some platforms, "ping6" would actually be used.

IPv6 Address Scopes

The scope of an address specifies in what part of the network it is valid and unique. The defined scopes in IPv6 are:

Node-local – valid only within the local node (e.g. loopback address)

Link-local – valid only within a single network link. All such addresses start with the ten bits "1111 1110 10" followed by 54 bits of 0 (fe80::/64). When specified in commands, you usually must follow a link-local address with "%" and the interface ID of the link it is connected to. In FreeBSD, this might be something like "fxp0", so to ping a link local address, you might use the command:

```
ping fe80::3c79:b2ca:90ce:5d59%fxp0
```

In Windows, interface IDs are numbers, so a ping command there might look like:

```
ping fe80::3c79:b2ca:90ce:5d59%11
```

Site-local – valid only within a "site". They start with the 10 bits "1111 1110 11" (fec0::/10). These were intended to be like IPv4 RFC 1918 "private addresses", but are no longer used as of RFC 3878, "Deprecating Site Local Addresses", September 2004.

Global – valid anywhere on the IPv6 Internet. Global unicast addresses are in the 2000::/3 block. When you specify global addresses, there is no need to append the interface ID, so a ping command for such an address might look like:

```
ping 2001:df8:5403:3000::c
```

IPv6 Address Types

A ***unicast address*** specifies a single network interface (destination address). Currently, all *global* unicast addresses are in the 2000::/3 block. There are also *link local* unicast addresses, in the fe80::/10 block. The global unicast address type is defined in RFC 3587 "IPv6 Global Unicast Address Format", August 2003. This RFC deprecates (makes historic) the "Top Level Aggregator" and "Next Level Aggregator" (TLA/NLA) scheme previously defined for global unicast addresses, and formalizes the 48-bit organization prefix, 16-bit subnet number and 64-bit interface identifier concept used today.

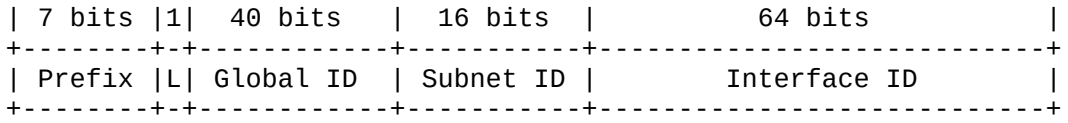
```
| 3 |     45 bits         |  16 bits  |       64 bits              |
+---+---------------------+-----------+----------------------------+
|001|global routing prefix| subnet ID |       interface ID         |
+---+---------------------+-----------+----------------------------+
```

There are two *special* unicast addresses:

:: (all bits zero) – the unspecified address, must never be assigned to any node
::1 (127 zeros followed by a 1) – the loopback address for IPv6 (corresponds to 127.0.0.1 in IPv4)

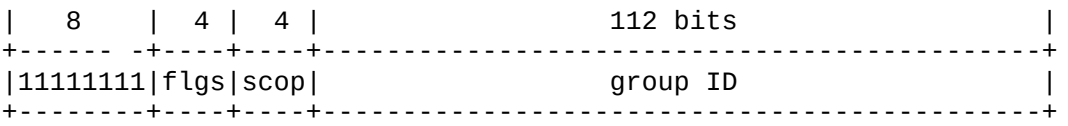
When Site-local scope was deprecated, a new address type called ***Unique Local Unicast*** was defined in RFC 4193, "Unique Local IPv6 Unicast Addresses", October 2005. These addresses are in the *fc00:/7* block. The first 7 bits are "1111 110". The 8th bit is called "L". If L = 1 the address is locally assigned (L = 0

is reserved for future use). The next 40 bits are a Global ID that insures the global uniqueness of the overall address. It is generated pseudo-randomly, and must not be sequential. The next 16 bits are a subnet ID and the final 64 bits are an interface ID (just like in global unicast addresses). Perhaps someday there will be a way to reserve specific Global IDs from a central authority (to prevent anyone else from using one you have chosen), but no such mechanism exists today. These addresses have much the same semantics as the IPv4 private addresses.



An ***anycast address*** can specify any of a group of addresses (usually on different nodes). A packet sent to an anycast address will be delivered to exactly one of those interfaces, typically the "nearest" one (in the network sense, not geographic sense). Anycast addresses look just like unicast addresses, and differ only in being injected into the routing protocol at multiple locations in the network.

A ***multicast address*** specifies multiple network destinations (multiple nodes can be configured with the same multicast address). A packet sent to a multicast address will be delivered to *all* nodes that have been assigned that address. Multicast addresses all have the special prefix *ff00::/8* (the first 8 bits of multicast addresses are all ones). After the first 8 bits, there are 4 bits of flags (0,0,0,T). If T=0, the address is a "Well Known" address assigned by IANA. If T=1, then the address is a non-permanently assigned ("transient") address. The scope is specified in the next 4 bits, followed by 112 bits of group ID:



There are several multicast scopes defined by the four scope bits. All other combinations are unassigned.

```
0  reserved
1  interface-local scope
2  link-local scope
3  reserved
4  admin-local scope
5  site-local scope
8  organization-local scope
E  global scope
F  reserved
```

The following multicast groups are "well known" (T=0):

```
1     node
2     router
5     OSPF IGP router
6     OSPF IGP Designated router
9     RIP router
a     EIGRP router
b     mobile agent
```

```
d     PIM router
16    MLDv2 capable router
fb    DNS server
101   NTP server
108   NIS+ server
1:2   DHCPv6 relay agent or server
1:3   DHCPv6 server (but not relay agent)
```

As there are 112 bits for *group* ID, there are 2^{112} (about 5.19 e+33) possible multicast groups. That is enough for the entire world, for quite some time to come. You can think of a multicast group as similar to a TV channel number. As examples, the following multicast addresses are all valid (and are all "well known"):

```
ff02::1           All nodes on the local link
ff05::1           All nodes in the organization

ff02::2           All routers on the local link
ff05::2           All routers in the site

ff02::fb          All DNS servers on the local link
ff08::fb          All DNS servers in the organization

ff02::1:2         All DHCPv6 relay agents or servers on local link
                  (note, DHCPv6 relay agents can only be reached
                  via link local addresses, so wider scope
                  addresses for relay agents don't make sense)

ff02::1:3         All DHCPv6 servers on the local link
ff05::1:3         All DHCPv6 servers in the site
```

With the scopes larger than the organization, multicast addresses must be specifically configured on nodes (you have to "subscribe to that channel"). If you ping the multicast address *ff0e::1*, you are not going to get a response from every node on earth, unless you can first talk everyone into adding that address to their nodes. Even then, various routers along the way would probably block *that* packet. An organization's routers enforce the scope rules so that link-local multicast addresses will not cross any routers, organization-local multicast addresses will not cross the organization's border router, but global multicast addresses will cross *any* router (in the real world, this is actually managed by the MLD – the Multicast Listener Discovery protocol and PIM, the Protocol-Independent Multicast protocol).

A ***solicited node multicast address*** is a special multicast address (addressed to all nodes on the local link) created from a global unicast address by appending the least significant (rightmost) 24 bits of the unicast address to the special prefix *ff02:0:0:0:1:ff::/104*. For the global unicast address

```
2001:df8:5403:3000:3c79:b2ca:90ce:5d59
```

the solicited node multicast address is:

```
ff02::1:ffce:5d59
```

These addresses are used by ND (the Neighbor Discovery protocol) in the process of mapping IPv6 addresses to Link Layer (MAC) addresses.

There is no ***broadcast address*** in IPv6, but a multicast to the *all nodes on the local link* multicast group *ff02::1* will have pretty much the same result.

Perhaps someday there will be a central authority to coordinate use (and allow reservation) of multicast group IDs. No such authority currently exists. Once IPv6 multicast broadcasters start making their programming available over large regions (or even worldwide), such coordination will be necessary, and corresponds to the FCC's management of broadcast frequencies that prevent stations from interfering with each other. Because the number of potential group IDs is so large (2^{112} or about 5.19 e+33), for now, choosing them randomly is sufficient. The probability of any two randomly generated group IDs being the same is quite low, even with millions of people using this scheme. You might think of these group ID's as being in some sense *channel numbers* as found today on TVs. I can envision a search engine that would allow you to find multicast channels associated with programming that caters to specific tastes, such as Bollywood Music Videos over IPTV.

Special Case: IPv4 Compatible IPv6 Addresses (Now Deprecated)

The entire 4.3 billion addresses of IPv4 are mapped into the IPv6 address space, not just once, but twice. Once as *IPv4 compatible IPv6 addresses* (::w.x.y.z), and a second time as *IPv4-mapped IPv6 addresses* (::ffff:w.x.y.z).

The addresses in the first special block all start with 96 bits of 0, followed by a 32 bit IPv4 address (which can be specified in dotted decimal). When you send traffic to an IPv4-compatible IPv6 address, it is sent as an IPv6 packet, but encapsulated with an IPv4 header, with the protocol field of the IPv4 packet header set to 41 to indicate that the payload is an IPv6 packet. The IPv4 header allows the traffic to travel across an IPv4-only infrastructure. Upon receipt, the packet payload (the IPv6 packet) is passed to the IPv6 protocol. This is called automatic IPv6 tunneling over IPv4 networks (defined in RFC 2893, "Transition Mechanisms for IPv6 Hosts and Routers", August 2000).

IPv4 compatible IPv6 addressees were deprecated in RFC 4291, "IP Version 6 Addressing Architecture", February 2006. No current transition mechanism uses them. New implementations are not required to support these addresses. Note however that two special addresses that are widely used actually fall into this range, the "unspecified" address (all zeros, or "::"), and the loopback address, ("::1").

Special Case: IPv4-Mapped IPv6 Addresses (Still Valid but Not Recommended)

The addresses in the second special block of addresses all start with 80 bits of 0 (*0:0:0:0:0*), followed by 16 bits of 1 (*ffff*), then a 32 bit IPv4 address (which can be, but does not have to be, specified in dotted decimal). When such an address is used on a dual stack node that supports IPv4-mapped IPv6 addresses, it causes an IPv4 packet to be sent using the last 32 bits of the IPv4-mapped IPv6 address, as the IPv4 address. As an example, on a Windows 7 node configured with dual stack, you can ping an IPv4 node as usual with the command:

```
C:\Users\lhughes>ping 10.1.0.14

Pinging 10.1.0.14 with 32 bytes of data:
Reply from 10.1.0.14: bytes=32 time<1ms TTL=64
Reply from 10.1.0.14: bytes=32 time<1ms TTL=64
```

```
Reply from 10.1.0.14: bytes=32 time<1ms TTL=64
Reply from 10.1.0.14: bytes=32 time<1ms TTL=64
```

You could ping the same IPv4, by using an IPv4-mapped IPv6 address, as follows. The ping command would first view the address as a valid IPv6 address, and create an IPv6 socket as usual. The IPv6 socket would look at the IPv6 address, realize it is an IPv4-mapped IPv6 address, then hand the operation over to the IPv4 stack to handle, using the low 32 bits of the IPv4-mapped IPv6 address. Normal IPv4 packets would be sent from the IPv6 socket, indistinguishable from the IPv4 packets sent in the example above.

```
C:\Users\lhughes>ping ::ffff:10.1.0.14

Pinging 10.1.0.14 with 32 bytes of data:
Reply from 10.1.0.14: bytes=32 time<1ms TTL=64
Reply from 10.1.0.14: bytes=32 time<1ms TTL=64
Reply from 10.1.0.14: bytes=32 time<1ms TTL=64
Reply from 10.1.0.14: bytes=32 time<1ms TTL=64
```

In general you can do any I/O operation to an IPv4 node using IPv4 packets, *from an IPv6 socket*, by using these IPv4-mapped addresses (on nodes where this is supported). Some operating systems (e.g. OpenBSD) don't support this kind of "cross-stack" operation at all. On some operating systems (Linux, NetBSD, FreeBSD) this mode is disabled by default, but can be enabled by including the following line in /etc/rc.conf:

```
…
IPv6_IPv4mapping="YES"
…
```

In general, it is best to avoid use of these addresses since support varies from operating system to operating system, behavior is implementation dependent, and there are potential vulnerabilities if it is enabled. It was originally intended as a transition mechanism, but it caused more problems than it solved, so it is better left unused, and ideally, disabled.

Simple IPv6 Address Assignment Scheme (for Manually Assigned Addresses)

The following is not part of any standard, IETF or otherwise. It is a best-practices recommendation, which may help you in migration to IPv6.

Many administrators have adopted a simple scheme for assigning IPv6 addresses manually to nodes, based on existing IPv4 address conventions or actual addresses. It could be argued that it can lead to confusion (by humans) between decimal and hexadecimal. It uses the same numeric digits that are currently used in your IPv4 scheme, to create what are really hexadecimal fields. It is possible to use the numeric digits (0 to 9) to create up to three hex digits in each of the four 16-bit groups in the IPv6 interface identifier. The resulting address may look strange in binary, but this scheme will make it easier for you to keep track of your IPv6 nodes, and is especially useful in dual stack networks, where you can use what appears to be the "same" address (not counting the prefix) on a given node, in both IPv4 and IPv6.

As an example, say our 48-bit organization prefix is 2001:df8:5403::/48. Let's also say we have four subnets (independent links) for IPv4, so we would also have four subnets for IPv6. Let's arbitrarily assign the IPv6 subnet numbers as 3000, 3100, 3200 and 3300 (all hex) for these subnets. Choose any values

you want for subnet numbers (when setting up your network architecture) – you have 65,536 (from 0000 to ffff) to play with. The following IPv4 addresses from these subnets could be assigned the corresponding IPv6 addresses:

IPv4 Subnet	IPv4 Address	Corresponding IPv6 Address
123.45.67.00/24	123.45.67.1	2001:df8:5403:3000:123:45:67:1
192.168.0.0/16	192.168.5.13	2001:df8:5403:3100:192:168:5:13
172.16.0.0/12	172.31.25.32	2001:df8:5403:3200:172:31:25:32
10.0.0.0/8	10.30.1.43	2001:df8:5403:3300:10:30:1:43

Alternatively, It is also possible to use just the *interface identifier* part of the IPv4 address ("node number within subnet")as the IPv6 interface identifier, in which case, the above addresses would be:

Subnet	IPv4 Address	Corresponding IPv6 Address
123.45.67.00/24	123.45.67.1	2001:df8:5403:3000::1
192.168.0.0/16	192.168.5.13	2001:df8:5403:3100::5:13
172.16.0.0/12	172.31.25.32	2001:df8:5403:3200::15:25:32
10.0.0.0/8	10.30.1.43	2001:df8:5403:3300::30:1:43

The mapping for the 172.31.25.32 address may confuse you – this is because a /12 subnet mask length divides the second 8 bit field right (31) in the middle (4 bits of it are network address, and 4 bits are interface identifier). This is why using dotted decimal for IPv4 was a bad idea, and hexadecimal is used in IPv6. This can get even more confusing with very odd subnet lengths, like /19. The following should clear things up:

172	31	25	32	Full address, dotted decimal
A C	1 F	1 9	2 0	Full address, hex
1010 1100 0001	1111 0001	1001	0010 0000	Full address, binary
	F	1 9	2 0	Interface identifier, hex
	15	25	32	Interface identifier, dotted decimal

Using only the IPv4 interface identifier is less likely to produce addresses that collide with automatically generated addresses, but requires good understanding of IPv4 subnetting (see above). Use whichever scheme makes the most sense to you, but try to be consistent.

The Simple IPv6 Address Assignment Scheme can also be used to manually assign link-local addresses. In this case, there is no IPv6 subnet number, because each address is valid only within a subnet. The following link-local addresses could be assigned to the above nodes:

Subnet	IPv4 Address	Corresponding IPv6 Address
123.45.67.00/24	123.45.67.1	fe80::123:45:67:1
192.168.0.0/16	192.168.5.13	fe80::192:168:5:13
172.16.0.0/12	172.31.25.32	fe80::172:31:25:32
10.0.0.0/8	10.30.1.43	fe80::10:30:1:43

As with global unicast addresses, you could use just the interface identifier part of each IPv4 address, which would result in the following manually assigned IPv6 link-local addresses:

Subnet	IPv4 Address	Corresponding IPv6 Address

```
123.45.67.00/24     123.45.67.1         fe80::1
192.168.0.0/16      192.168.5.13        fe80::5:13
172.16.0.0/12       172.31.25.32        fe80::15:25:32
10.0.0.0/8          10.30.1.43          fe80::30:1:43
```

Note that the address 123.45.67.1/24 and 192.168.0.1/16 would both produce fe80::1 as the equivalent IPv6 address, but this would not produce a conflict since they are in different subnets, and link-local addresses are valid only within a single subnet.

Obviously no addresses generated with Stateless Address Autoconfiguration will use this convention, although you should be careful to make sure there are no conflicts between addresses you create and automatically generated addresses. Duplicate Address Detection during automated address creation should detect such conflicts. On the other hand, you can easily create DHCPv6 address pools that will be consistent with these schemes.

Warning: There is a perfect valid (but not often used) textual representation of IPv6 addresses that would allow you to use the exact same bits as a 32 bit IPv4 interface identifier, and even specify those 32 bits in dotted decimal. However, it mixes hexadecimal and decimal numbers, plus colons and dots in a single address representation, which to me is extremely confusing and inelegant. It represents the first 96 bits of an address in *coloned hex* notation, and the last 32 bits of that address in *dotted decimal* notation. When you use this *mixed* notation, you must always specify all four dotted decimal fields, and they must be the least significant 32 bits. It is possible that some software applications will not accept this representation. Also, many things that report addresses (e.g. ipconfig) have no way to know to display some addresses in mixed notation and others in regular coloned hex notation, so they just display all addresses in coloned hex notation. This can lead to confusion. As examples of addresses with this mixed notation, the above IPv4 addresses would have corresponding IPv6 addresses that look like this:

IPv4 Address	IPv6 Address in “Mixed” Notation	Same Address in Coloned Hex
123.45.67.1	2001:df8:5403:3000::123.45.67.1	2001:df8:5403:3000::7b2d:4301
192.168.5.13	2001:df8:5403:3100::192.168.5.13	2001:df8:5403:3000::c0a8:50d
172.16.25.3	2001:df8:5403:3200::172.16.25.3	2001:df8:5403:3000::ac10:1903
10.30.1.43	2001:df8:5403:3300::10.30.1.43	2001:df8:5403:3000::a1e:12b

I recommend that you avoid use of this mixed notation altogether. If you use the Simple IPv6 Address Assignment scheme, be very careful to use colons (not dots) between all fields, as software that understands the mixed address syntax will interpret addresses with dots in the last 4 groups as perfectly valid “mixed” notation. This will result in some odd problems. The mixed notation was really intended for use with IPv4-mapped IPv6 addresses, but it works anywhere. You should never create addresses using it, but you need to know about it in case you see addresses written in it by someone else.

Multiple IPv6 Subnet Numbers on a Single Network Link

A single network link can actually have addresses with more than one 16 bit subnet number at any given time. For example, the prefix 2001:df8:5403:1600::/64 may be used with stateless auto configuration, while the prefix 2001:df8:5403:1601::/64 could be used with stateful auto configuration using DHCPv6 on the same network link. You could also have manually assigned addresses using a *third* prefix (e.g. 2001:df8:5403:1602::/64) on the same network link. Addresses with different subnet numbers, but the same interface identifier, *are not in conflict*. Normally, you only broadcast one 64-bit prefix with Router

Advertisement messages onto a given network link, so all address created with stateless auto configuration in a given subnet will have only that one 64-bit prefix. It is possible in some implementations to advertise up to 100 prefixes on each network link. If multiple prefixes are advertised, there will still be only one default gateway, which is the link-local address of the gateway that is sending Router Advertisement messages. Another alternative is to define a subnet size *greater than /64* on a single network link that includes all of the desired subnet numbers. With a "/60" subnet, you can actually have 16 sequential /64 subnet numbers in a single network link (the first subnet number has to be an integral multiple of 16). This is called *supernetting*.

Multiple IPv6 Addresses on a Single Node

Unlike with IPv4, it is normal for IPv6 nodes to have *multiple* valid addresses. They don't even all have to have the same subnet number (if you are running multiple subnet numbers on a single link). A single node could have addresses with each of the above 64-bit prefixes (or even multiple manually assigned addresses) at any given time. It could also have various multicast addresses. One of the unicast addresses (chosen at random) will be used as the source address of packets sent by that node, but incoming packets addressed to any of the addresses owned by the node will be accepted.

A host is required to recognize any of the following addresses as referring to itself. Any node has most of these *by default* without anyone having to assign them. The default link-local address is created with Stateless Address Autoconfiguration even if there are no Router Advertisement messages. *Solicited-Node Multicast* addresses are created and assigned automatically when unicast or anycast addresses are assigned.

- The Loopback Address (::1) – always present.
- The All-Nodes Multicast Addresses (ff01::1, ff02::1, etc.) – only the "on node" and "on link" scoped multicast addresses are created automatically – ones with larger scope must be specifically assigned to each node that you wish to accept such addresses.
- The automatically generated link-local unicast address.
- Any additional Unicast and Anycast Addresses that have been assigned to any of the node's interfaces, manually or automatically.
- The Solicited-Node Multicast Address for each of its unicast and anycast addresses (created automatically for you when the corresponding unicast or anycast address is assigned).
- Multicast Addresses for all other groups to which the node has subscribed.

A router (gateway) is required to recognize all addresses that a host is required to recognize, plus the following special addresses for routers, as identifying itself:

- The Subnet-Router Anycast Address for all interfaces for which it is configured to act as a router.
- All other Anycast Addresses with which the router has been configured.

- The All-Routers Multicast Addresses (ff01::2, ff02::2, ff05::2).

Automatically Generated Interface Identifiers based on EUI-64

By default, Stateless Address Autoconfiguration will create a link-local address (fe80::w:x:y:z). If there is a Router Advertisement daemon configured and running on the link, the node will also automatically create a global unicast address by using the 64-bit subnet prefix form the Router Advertisement message. It can generate the interface identifier (low 64 bits) either from the node's MAC address (using EUI-64), or can use a random 64 bit value. This is described in RFC 4291, "IP Version 6 Addressing Architecture" and RFC 2464 "Transmission of IPv6 Packets over Ethernet Networks".

An EUI-64 address is created by taking the *first 24 bits* of a MAC address (the *Organizationally Unique Identifier* part of the MAC address), setting the 7th bit of this to 1 (counting rightward from the Most Significant Bit), appending the 16 bit value FFFE, then appending the last 24 bits of the MAC address (the *device identifier*). Hence, the 48 bit MAC address

```
00-18-8B-78-DA-1A
```

produces an EUI-64 identifier of

```
0218:8BFF:FE78:DA1A
```

This is a reversible mapping, so given an EUI-64 identifier, it is trivial to determine the MAC address of the node (discard the FFFE in the middle 16 bits and clear the 7th bit of the remaining 48 bit value). Note: the 7th bit in the first byte of all valid Organizationally Unique Identifiers, hence of all MAC addresses, will always be 0.

One of the security advantages of IPv6 is supposed to be that the number of possible addresses in a subnet (2^{64}) is so large that it is impractical to scan all of them to discover all of the nodes on a subnet (this is called *mapping* a subnet). If EUI-64 interface identifiers are used, there are so few of these (in comparison to the total possible number of interface identifiers) that it *is* possible to scan for them (especially with the knowledge of which Organizationally Unique Identifiers are actually in use, which is not difficult to determine).

Randomized Interface Identifiers

There are several privacy concerns related to using addresses with EUI-64 interface identifiers. One is the ability for a hacker to create a map of all nodes on the subnets via scanning. It would also be possible to identify any person's traffic at any point through which the traffic flows, if you know the MAC address of their network interface. You could certainly associate various traffic flows that all have the same MAC address as coming from a single node. Normally MAC addresses never leave your LAN. With EUI-64 based IPv6 unicast addresses, MAC addresses can go anywhere in the world. Fortunately, there is a way to generate a random interface identifier instead of using the EUI-64 identifier. This is defined in RFC 4941, "Privacy Extensions for Stateless Address Autoconfiguration in IPv6", September 2007. The randomized identifier even changes automatically over time. I may have had *that* address

yesterday, but today I've got a completely different one! Interface identifier randomization is enabled by default in Windows 7, but it can be enabled or disabled with the following commands:

```
netsh interface IPv6 set global randomizeidentifiers=enabled
netsh interface IPv6 set global randomizeidentifiers=disabled
```

The reason you might want to disable randomization is that some servers will only accept a connection from nodes for which they can perform a reverse DNS lookup. This often will fail with randomized identifiers. Note that use of randomized interface identifiers can make it very difficult to determine to whom specific traffic in a log belongs, unless a record is kept of randomized interface identifiers used by each node.

When a randomized address changes, the old address is kept around for some time, but marked as *deprecated*, which means your node will not use it for further outgoing connections. You should accept incoming replies addressed to a deprecated address until it becomes *invalid*, which it eventually will be. Since you aren't making new outgoing connections with it, replies to it will cease fairly quickly. Addresses with randomized interface identifiers are used primarily for outgoing connections (and replies thereto). A node that can accept incoming connections from anyone should have (possibly in addition to other addresses) a static (unchanging) unicast address which is published in DNS. This would be used by other nodes that want to connect to it. A node that only ever makes outgoing connections need not have such a static address assigned to it, and there is no need to publish its name and IPv6 address in DNS (at least not in your *external* DNS). Remember in IPv6, it is *much* more likely that other nodes will be connecting to your node (for VoIP, VPNs, P2P, etc.). The age of NAT (and one-way connectivity) is over.

IPv6 Address Allocation

The standard allocation block to be given to organizations is a "/48", which is 65,536 subnets, each of which is a "/64" block consisting of 2^{64} or about 18 billion, billion addresses (about 4 billion times the total number of addresses in the First Internet). Some ISPs may choose to allocate only a single "/64" block to individuals or home users, who have no need for multiple subnets. It is not practical to allocate only a single IPv6 address (a" /128" block) to a user, due to the fact that nodes often create new addresses. One "/48" block will supply 65,536 individuals or homes with "/64" blocks. Perhaps I'm a bit unusual, but I already have two subnets in my home today (one dual stack, one IPv6-only). Who knows, I might have a bunch someday! My company has a "/48" (2001:df8:5403::/48) which we divided into 16 "/52" sub-blocks, each of which have 4096 subnets. I have one of these "/52" sub-blocks (subnets 3000 to 3fff) routed to my house. That should just about take care of me for some time to come. A single "/64" block should work for most home users.

ISPs are allocated *really* big "/32" blocks of addresses, which are enough to allocate "/48" blocks for up to 65,536 customers. Should they use up an entire "/32" block, there are *plenty* more "/32" blocks where that one came from (about 536 million of them just in the 2000::/3 block marked for allocation). The RIR's (ARIN, RIPE, APNIC, LACNIC and AfriNIC) will be happy to give an ISP all they can use. If you assume there are 7 billion people alive, there are over 5000 "/48" blocks for every human alive, just out of the 2000::/3 range currently marked for allocation. It is extremely unlikely that *any* single human will ever be able to use any appreciable percentage of their "fair share" of addresses, let alone have the

IANA run out. The folks in Taiwan say they want to connect 3 billion devices to the Internet in the next couple of years. This would take ¾ of the entire First Internet's address space, but could be handled with a tiny fraction (less than 1 billionth) of a single "/64" block with IPv6, should they want to have them all in one block for some bizarre reason. It will be quite a while before anyone worries about IPv6 address space exhaustion (famous last words?).

The People's Republic of China believes that they were cheated out of sufficient IPv4 addresses to participate fully in the First Internet. By the time China started deploying TCP/IPv4, if they had taken *all* of the remaining addresses, over 90% of the people there would not have gotten one. The First Internet recently passed an interesting threshold. There are now more Chinese speaking users on it than English speaking users. If you recall the chart of allocated addresses in section 3.3.2.2 of this book) the U.S. has over 43% (28% ARIN + 15% Legacy, both of which are mostly U.S. users) of the total IPv4 address space for less than 5% of the world's population. In comparison, APNIC, which includes China, India and several other populous countries (all together about 50% of the world's population), has only 16% of the IPv4 address space. When the IPv4 addresses are all gone in September 2011, APNIC will probably still have less than 20% of the IPv4 address space (about .28 addresses per person), while the U.S. will probably have about 45% (about 6.4 addresses per person). However, note that about 1/3 of that 45% are held by less than 50 organizations (like M.I.T, Apple, HP, etc.). The distribution of addresses in the First Internet was (and remains) anything but equitable. It's really pretty much impossible to do anything about that now. We're doing it right in the Second Internet.

Should We Reserve Some IPv6 Addresses for Developing Nations?

There has been talk from the ITU (International Telecommunication Union) about reserving some IPv6 address space for developing nations to make *absolutely* certain that nobody ever gets left out again, as has happened in the First Internet. There are *so many* IPv6 addresses that there is essentially no chance of this ever happening. The ITU might as well try to reserve a few trillion grains of sand (maybe a dump truck's worth) to make sure that every country can be assured of getting their fair share of them. The total number of IPv6 addresses is on the same general scale as the number of grains of sand on Earth.

Note that block 2000::/3 (which you can also think of as blocks 2000::/16 through 3fff::/16) is currently the only part of the overall space marked for unicast address allocation. This is only 1/8 of the total IPv6 address space. Even so, this is still 2^{125}, or about 4.15 e+37 addresses. You can also view this as 2^{45} (about 35.2 trillion) "/48" blocks, or just over 5000 "/48" blocks *per human alive* in 2010 (using worldwide population as 7 Billion). Should we ever use this up, there is still at least 5.5 times that much space not used for anything (from 4000::/16 to efff::/16) that we could allow for additional allocation.

I personally don't think there is any reason to reserve a special block of addresses for anyone, including developing nations. Unlike with IPv4, there are plenty of addresses for everyone this time around.

The People's Republic of China (and every other country) will have *plenty* of addresses in the Second Internet, and this is one reason they are investing so heavily in it. India is now determined to deploy IPv6 nationwide, and should have quite a bit deployed by the end of 2010. The inequitable distribution of addresses in the First Internet may also account for some of the lack of urgency to migrate to the Second Internet in the United States. Unfortunately, it is not simply of matter of still having enough IPv4 addresses. Imagine if the U.S. stayed with Standard Definition NTSC TV, while the entire rest of the world went with globally standard High Definition TV. The U.S. would not be able to export their

programming to anyone else, nor import programming from the rest of the world. If they choose to stay with IPv4, they will be isolating themselves in some very serious ways. It's not completely ridiculous to think that the U.S. might decide not to deploy the Second Internet. Look what happened with the metric system. If IPv4 is "riding horses" and IPv6 is "driving cars", you don't need to wait until the last horse dies before you get a car. The "cars" (IPv6) are ready and widely available today. Those who adopt cars first will leave those still riding horses *way behind*. I'd suggest you migrate to IPv6 *as soon as possible*. Countries that master it and start creating products and applications based on it will have a giant head start in the 21st century over those who wait until the last possible minute.

How Is the Entire IPv6 Address Space Divided Up?

Here are the official allocations of the IPv6 address space as of 13 May 2008 (from IANA), along with the RFCs that allocated the blocks listed:

IPv6 Prefix	Allocation	Reference	Note
0000::/8	Reserved by IETF	[RFC4291]	[1] [5]
0100::/8	Reserved by IETF	[RFC4291]	
0200::/7	Reserved by IETF	[RFC4048]	[2]
0400::/6	Reserved by IETF	[RFC4291]	
0800::/5	Reserved by IETF	[RFC4291]	
1000::/4	Reserved by IETF	[RFC4291]	
2000::/3	Global Unicast	[RFC4291]	[3]
4000::/3	Reserved by IETF	[RFC4291]	
6000::/3	Reserved by IETF	[RFC4291]	
8000::/3	Reserved by IETF	[RFC4291]	
A000::/3	Reserved by IETF	[RFC4291]	
C000::/3	Reserved by IETF	[RFC4291]	
E000::/4	Reserved by IETF	[RFC4291]	
F000::/5	Reserved by IETF	[RFC4291]	
F800::/6	Reserved by IETF	[RFC4291]	
FC00::/7	Unique Local Unicast	[RFC4193]	
FE00::/9	Reserved by IETF	[RFC4291]	
FE80::/10	Link Local Unicast	[RFC4291]	
FEC0::/10	Reserved by IETF	[RFC3879]	[4]
FF00::/8	Multicast	[RFC4291]	

Notes:

[0] The IPv6 address management function was formally delegated to IANA in December 1995 [RFC1881].

[1] The "unspecified address", the "loopback address", and the IPv6 Addresses with Embedded IPv4 Addresses are assigned out of the 0000::/8 address block.

[2] 0200::/7 was previously defined as an OSI NSAP-mapped prefix set [RFC4548]. This definition has been deprecated as of December 2004 [RFC4048].

[3] The IPv6 Unicast space encompasses the entire IPv6 address range with the exception of FF00::/8. [RFC4291] IANA unicast address assignments are currently limited to the IPv6 unicast address range of 2000::/3. IANA assignments from this block are registered in the IANA registry: iana-IPv6-unicast-address-assignments.

[4] FEC0::/10 was previously defined as a Site-Local scoped address

```
     prefix. This definition has been deprecated as of September 2004
     [RFC3879].

[5]  0000::/96 was previously defined as the "IPv4-compatible IPv6
     address" prefix.  This definition has been deprecated by [RFC4291].
```

The referenced RFCs are:

- RFC 1881 – “IPv6 Address Allocation Management”, December 1995 (obsoleted by 4048)
- RFC 3879 – “Deprecating Site Local Addresses”, September 2004 (affects FEC0::/10)
- RFC 4048 – “RFC 1888 is Obsolete”, April 2005 (dropping mapping of OSI addresses)
- RFC 4193 – “Unique Local IPv6 Unicast Addresses”, October 2005
- RFC 4291 – “IP Version 6 Addressing Architecture”, February 2006

The 6bone was an early worldwide IPv6 testbed. It used addresses from 3ffe::/16 (as per RFC 2471, “IPv6 Testing Address Allocation”, December 1998). These have since been returned to the overall allocation pool as per RFC 3701, “6bone (IPv6 Testing Address Allocation) Phase-out”, March 2004, once the 6bone had served its purpose and was shut down. Interestingly, some addresses from this block still show up on the IPv6 backbone. Among other places, they are still used in IPv6 Ready tests, so if an IPv6 Ready test network is connected to the main Internet, those addresses could be accidentally routed.

As of January 2010, the RIR’s have the following number of IPv6 prefixes that actually have traffic on the backbone:

RIPE	1998
ARIN	1207
APNIC	852
LACNIC	267
AfriNIC	82

Here are the top ten countries plus a few from Asia (from SixXS, 24 Jan, 2010) ranked by number of IPv6 prefixes allocated. “V” means visible (actual traffic detected), “A” means allocated (obtained from an ISP or RIR), and “VP” is the percentage of all allocated blocks that are visible (total for the world would be 100%).

Rank	Country	V	A	VP
1	United States	422	1143	9.30%
2	Germany	179	324	3.24%
3	United Kingdom	100	225	2.20%
4	Netherlands	102	176	2.25%
5	Japan	93	176	2.05%
6	Australia	41	152	0.90%
7	Russia	54	117	1.19%
8	France	49	111	1.08%
9	Brazil	29	106	0.64%
10	Switzerland	56	102	1.23%
19	Korea	15	58	0.33%
20	China	21	54	0.46%

```
24    India                7    36  0.15%
31    Taiwan              19    33  0.42%
33    Vietnam              4    28  0.09%
34    Philippines          8    27  0.18%
35    Thailand            12    27  0.26%
```

Note that this data does not reflect actual number of addresses or the volume of traffic, just the number of distinct 48 bit prefixes, which is a rough indication of the *number of organizations* investigating IPv6. Much of this in the U.S. is probably research or academic. As percentages of the gigantic total number of "/48" blocks available for allocation, all of these are essentially zero (pretty much *all* of the 2000::/3 IPv6 address space is still available for allocation). This tiny percentage is more an indication of the colossal size of the IPv6 address space than of any lack of interest or activity.

Classless Inter-Domain Routing (CIDR)

There is no reason to define CIDR for IPv6, because it was done in IPv4 *only* to extend the lifetime of the IPv4 address space long enough for IPv6 to be fully developed, which has now happened. There is no need to extend the lifetime of the IPv6 address space. If IPv6 had been ready, and we had migrated to it in the mid 1990s, we would never have had to suffer through the complexities brought about by CIDR and NAT. The reason we are having to deal with these issues today is that we have already stayed with IPv4 *far too long*. Imagine trying to do serious work today with an 8 bit processor and 64K bytes of RAM.

Network Ports

Network ports work exactly the same way under IPv6 as they do in IPv4. There are still 65,536 of them associated with every IPv6 address. They could have gone to 32 bit port numbers (yielding 4.3 billion ports for each address), but this would have required even more changes in packet headers and other places, so this was not done. 65,536 is plenty for almost any need, especially since you can assign any number of global unicast addresses to a single interface (each of which has 65,536 ports). The same *Well Known Port* numbers are used in IPv6. The only difference is that you will never see port numbers on IPv6 addresses being shifted by a NAPT gateway, since there is no NAT for IPv6 to IPv6. Note that a given port being used over IPv4 does not prevent it from being used by the same or even a different application, over IPv6 (and vice versa).

5.3.2.3 – Subnetting in IPv6

There is no CIDR in IPv6 (although the CIDR "slash notation" is still used). As a result, subnetting is much simpler in IPv6. All subnets are "/64". The only exception is if you do *supernetting* (e.g. a "/60" subnet) to allow multiple "/64" blocks to be used on a single network link. This will likely only be done in large, advanced corporate networks, so most network engineers will never see anything but "/64" subnets. The only reason for doing this might be to use different "/64" subnets for specific purposes, such as 1000 for SLAAC, 1001 for DHCPv6 assigned addresses and 1002 for manually assigned addresses. If you use EUI-64 interface identifiers for SLAAC, it is not difficult to partition a single "/64" so there will be no overlap between SLAAC, DHCPv6 and manual assignments. If you use random interface identifiers, they may fall anywhere a "/64" address space. However, the probability of one colliding with an address assigned manually or via DHCPv6 stateful mode is incredibly low, and Duplicate Address Detection

should prevent the odd collision. Having at least two "/64" subnets in a single network (one for SLAAC, one for manual and DHCPv6 assigned addresses) removes *all* possibility of an address collision.

Each subnet needs to be *at least* a "/64", since EUI-64 can generate "node within subnet" values that are 64 bits long. Randomized interface identifiers are also 64 bits in length. But a "/64" subnet is *already larger* than any organization could conceivably use (18 billion, billion addresses). There are so many "/64" blocks in a single "/48" (65,536) that we can use them even for subnets between a border router and a firewall, which have only two addresses. There is never an excuse to use any subnet smaller than a "/64", although I have seen some old-school IPv4 trained administrators allocate "/124" IPv6 subnets for the link between a border gateway and firewall case (in IPv4, tiny subnets like /30 would be used in such a case). Old habits die hard. After living with increasing scarcity with IPv4 addresses, it is hard for some of us to realize that there are PLENTY of addresses this time around.

5.3.2.4 – Link Layer Addresses

The software in the *Application Layer*, the *Transport Layer*, and the *Internet Layer* of the TCP/IPv6 stack think in terms of IP addresses. But the *Link Layer* (and the hardware) thinks in terms of MAC addresses. In IPv6 the mapping from IPv6 address to Link Layer (MAC) address is done with the Neighbor Discovery protocol. Note that in this book, I often use the terms *Link Layer Address* and *MAC address* interchangeably.

NOTE: A Link-Layer address is a "MAC address" only for Ethernet based network hardware (and a few others), so when I use the term MAC address, think "physical layer address for the actual network hardware in use". The term Link-Layer address is more accurate (a MAC address is just a special case of *Link Layer Address*), but it is easy to confuse it with the similar sounding term *link-local address*. Just realize that if the actual network in use is not Ethernet there may be some other name for the physical layer addresses that IP addresses have to be mapped onto, and it may not look anything like the 48 bit MAC address.

IPv6 addresses are not actually used at the lowest layer of the TCP/IPv6 network stack (the *Link Layer*). The 48 bit MAC addresses covered in the TCP/IPv4 chapter still exist and are used the same way at the *Link Layer* (at least for Ethernet networks).

Neighbor Discovery Protocol (ND)

There is no ARP (Address Resolution Protocol) in IPv6. The new ND (Neighbor Discovery) protocol which is defined in RFC 4861 "Neighbor Discovery for IP version 6 (IPv6)", September 2007, accomplishes the same thing, and many other functions as well, including:

- *Router Discovery*: hosts can locate router(s) residing on any link to which it is attached.
- *Prefix Discovery*: hosts can discover the correct 64 bit prefix for any link to which it is attached.
- *Parameter Discovery*: hosts can determine the correct IPv6 parameters, for any link to which it is attached, such as MTU.

- *Stateless Address Autoconfiguration* (SLAAC): hosts can automatically obtain a *link local* address; and, if a Router Advertisement Daemon exists, also a *global unicast* address.

- *Address Resolution*: mapping IPv6 addresses to MAC addresses (as the replacement for ARP).

- *Next-hop Determination*: hosts can determine next-hop router for a given destination address.

- *Neighbor Unreachability Detection* (NUD): determine that a given neighbor is no longer reachable on any attached link (there is no corresponding IPv4 functionality).

- *Duplicate Address Detection* (DAD): hosts can determine if a proposed address is already in use.

- *Redirect*: router can inform a host about a better (or working) first-hop.

There are five ICMPv6 messages that ND uses to accomplish these things:

- *Router Solicitation:* Request a Router Advertisement message.

- *Router Advertisement*: Router advertises the 64-bit prefix and parameters for a link, usually sent by a Router Advertisement Daemon living in a gateway router or firewall. The Router Advertisement Daemon can send different information into each attached link, if there are multiple links. This also tells nodes whether or not there is a DHCPv6 server available.

- *Neighbor Solicitation:* Any node can say "howdy neighbor" to another node to see if it responds.

- *Neighbor Advertisement:* Response to a "howdy neighbor" message from someone else.

- *Redirect*: A router can inform any node that there is a better first-hop available than one it has just tried ("there's a bridge out along *that* road, try going down *this* road"), based on its discovered knowledge of the surrounding network.

By the way, some people use "NDP" as the acronym for the Neighbor Discovery protocol (see Wikipedia). If you read the RFCs, the creators of the protocol use just "ND", so we will use that convention in this book. The acronyms of some protocols include the "P" (for Protocol) in the acronym (e.g. TCP), while others don't (like MLD). I follow the conventions used in the RFCs.

IPv6 Router Advertisement messages carry link-layer (MAC) addresses, so no additional packet exchange is required to resolve the router's link-layer address. They also carry prefixes, so no separate mechanism is needed to configure a netmask.

By using link-local addresses to uniquely identify routers, hosts can maintain router associations. This capability is necessary for Router Advertisements, and for redirects. Hosts need to maintain router associations if the site switches to a new global prefix.

ND is immune to spoofing attacks that originate from off-link nodes. In IPv4, off-link nodes can send ICMPv4 Redirect messages, and IPv4 Router Advertisement messages.

In the following, DAD refers to *Duplicate Address Detection*, which is one of the functions performed by ND. Addresses may be in any one of the following states at any given time:

- TENTATIVE: Generated, but not yet determined by DAD to be unique – can be used only for sending and receiving ND messages for DAD.

- DUPLICATED: Generated, and determined by DAD to be duplicated (hence unusable).

- PREFERRED: Generated , and determined by DAD to be unique (hence valid).

- DEPRECATED: A preferred address that has passed its *preferred lifetime* (still valid, and incoming packets addresses to it will be accepted, but no further outgoing packets will be sent using it).

- INVALID: A deprecated address that has passed its *valid lifetime* (may no longer be used for sending *or* receiving packets).

Here are the details of the various functions that ND can perform:

Router Discovery – at any time (but typically at power on), any node can determine the link-local address of the router(s) on the local link.

> Step 1 – the node sends a Router Solicitation message to the "all routers on link" multicast group (*ff02::2*). If the node's link local address has already been created, then that will be used as the source address, else the unspecified address ("::") will be used as the source address.
>
> Step 2 – All routers on the link will respond with Router Advertisement messages, usually to the "all nodes on link" multicast group (*ff02::1*), but if the source address of the Router Solicitation message was a link-local address the router can choose to send the Router Advertisement message directly to that address. The source address of each received Router Advertisement message is added to a default gateway table (from which the preferred link-local default gateway will be chosen). The Prefix Information option in all of the responses should be the same, so the subnet prefix from the last received Router Advertisement message will be used.

IPv6 Router Discovery corresponds roughly to IPv4 Router Discovery (which was defined in RFC 1256, "ICMP Router Discovery Messages", September 1991), but in IPv6 it is a part of the base protocol. There is no need for hosts to *snoop* the routing protocols to discover a router. IPv4 router discovery contains a preference field, which is not needed in IPv6 Router Discovery because of Neighbor Unreachability Detection. IPv4 Router Advertisements and Solicitations (ICMP type 9) work only with multicast capable IPv4 routers, and are not commonly used. All IPv6 nodes support multicast, and Router Advertisements are a fundamental part of almost every non-trivial network.

Address Resolution **(Mapping IPv6 addresses to MAC addresses)**

Say Alice (one IPv6 node) is trying to send a packet to Bob (another IPv6 node). Address resolution is done as follows:

Step 1 – Alice checks her *Neighbor Cache* (similar to the ARP table in IPv4) to see if it already has an entry with Bob's IPv6 address. If it does, then Alice sends the packet immediately to Bob using Bob's MAC address from her Neighbor cache, and she is finished. If Alice's Neighbor Cache doesn't have an entry for Bob's IPv6 address, the process continues.

Step 2 – Alice adds a new Neighbor Cache entry for Bob, in the INCOMPLETE state. Alice then sends a Neighbor Solicitation message to Bob, using Bob's *solicited-node multicast address* as the destination address. Any of the addresses assigned to Alice's interface can be used as the source address of this packet, but if possible it should match the source address of the original packet Alice wanted to send. Alice includes her MAC address as the Source Link-layer Address option in this packet. This insures Bob will have her MAC address when it's time for him to reply.

Step 3 – Bob receives the Neighbor Solicitation message, and responds with a Neighbor Advertisement message, sent to Alice's MAC address.

Step 4 – Alice receives the Neighbor Advertisement message from Bob, and then updates Bob's entry in her Neighbor Cache

Step 5 – Alice can now send the original packet she wanted to send to Bob using his MAC address.

Prefix Discovery

At any time, a node can discover the default network prefix. A Router Advertisement message can contain up to three "options":

- the Source Link-Layer Address (the sending router's MAC address).
- the MTU (the maximum packet size supported on this link).
- the Prefix Information (the preferred address prefix for this subnet).

When a router sends an unsolicited Router Advertisement message, it includes all three options. In a solicited Router Advertisement message, at least the Prefix and MTU options will be included, so in either case, the node will obtain the preferred prefix for the link.

Step 1 – The node wanting to discover the subnet prefix sends a Router Solicitation message, using its own link-local address as the source, and the "all routers in local link" multicast group (*ff02::02*) as the destination address.

Step 2 – All routers on the local link respond with Router Advertisement messages, with their own link-local address as source and the "all nodes on local link" multicast group (*ff02::1*) as the destination. The Router Advertisement message includes at least the subnet prefix option. This prefix is extracted from the prefix option, and stored as the subnet prefix. All routers will respond with the same prefix, but the last Router Advertisement message received will have the subnet prefix that is used.

Duplicate Address Detection (DAD)

DAD is used to determine if a proposed (tentative) address is a duplicate of any address on the local link. Both hosts and routers perform DAD on *all* unicast and anycast addresses regardless of how they are obtained (Stateless Address Autoconfiguration, DHCPv6 or even manual assignment). DAD is accomplished using Neighbor Solicitation and Neighbor Advertisement messages.

> Step 1 – The node owning the tentative address sends a number of Neighbor Solicitation messages using the unspecified address (::) as the *source address*, the Solicited-node multicast address as the *destination address* and the TENTATIVE address as the *target address*.
>
> Step 2 – If any node on the link is already using the TENTATIVE address, it will respond by sending a Neighbor Advertisement to the "all nodes on local link" multicast group (*ff02::1*). If no such response is seen during a short interval (configurable), then the TENTATIVE address is considered to be unique.

Stateless Address Autoconfiguration (SLAAC)

This is one of the most important new aspects of IPv6. It is primarily used to allow IPv6 capable hosts (as opposed to routers) to automatically obtain address information (link local and global unicast node addresses and link-local default gateway). Routers use it to generate and validate their link local addresses (but not their global addresses, which must be statically configured). The process makes strong use of link-local and multicast addresses, and all network communication is done with ICMPv6 messages that are part of ND. If a source of Router Advertisement messages (e.g. a router or firewall) is available, then at least one global unicast IPv6 address will also be generated. The acronym for Stateless Address Autoconfiguration is "SLAAC".

There are four steps involved in Stateless Address Autoconfiguration:

> Step 1 – the node creates a 64-bit interface identifier. This can be created using the MAC address and the EUI-64 algorithm, or can be a randomly generated value ("randomized interface identifier").
>
> Step 2 – the host creates a TENTATIVE link-local address. This is done by appending the chosen interface identifier to the prefix *fe80://10*. DAD is performed to determine if the link-local address is unique. If so, that address goes to the PREFERRED state, its lifetime starts counting, and the process continues. If the address is duplicated the address goes to the DUPLICATED state, the interface is disabled, and the SLAAC process fails without having generated any addresses.
>
> Step 3 – the host sends a Router Solicitation message to the "all routers on link" multicast group (*ff02::2*). If the node's link local address has already been created, then that will be used as the source address, else the unspecified address ("::") will be used as the source address. All routers on the link will respond with Router Advertisement messages, usually to the "all nodes on link" multicast group (*ff02::1*), but if the source address of the Router Solicitation message was a link-local address the router can choose to send the Router Advertisement message via unicast to

just that address. The source address of each received Router Advertisement message is added to a default gateway table (from which the preferred link-local default gateway will be chosen). The Prefix Information option in all of the responses should be the same, so the subnet prefix from the last received Router Advertisement message will be used.

If no router responds to the Router Solicitation message within a certain time, then the SLAAC process terminates, having created a valid link-local node address, but no link-local default gateway and no global unicast address.

Step 4 – if we reach this step, a valid Router Advertisement was received with a subnet prefix, so the host combines the discovered subnet prefix with the created interface identifier, to create a TENTATIVE global unicast address for the node. DAD is performed on the tentative global unicast address, and if the address is unique, it goes to the PREFERRED state and its lifetime starts counting. If not, the address goes to the DUPLICATED state, the interface is disabled, and the SLAAC process terminates, again having created a valid link-local address and a link-local default gateway address (but no global unicast address).

Anytime a link-local or global address lifetime expires (enters the INVALID state), address regeneration is done. If using randomized interface identifiers, a different random interface identifier is created for each address regeneration. If using EUI-64 interface identifiers, the regeneration process basically just confirms that the addresses are still valid – they don't actually change. If something has changed since the last validation (e.g. gateway down, link broken, etc.), the SLAAC process may fail and the address is marked INVALID.

Next-hop Determination

When one node needs to send a packet to another node, the sending node must determine whether the destination address is *on-link* or *off-link*. To be considered *on-link*, the address must match at least one of the following criteria:

- The prefix of the address must match one of the prefixes assigned to the link.
- The address is the target of a Redirect message sent by a router.
- The address is the target address of a Neighbor Advertisement message.
- The address is the source address of any Neighbor Discovery message received by the node.

If the address is *on-link* then the next-hop address is the same as the destination address. If the address is *off-link* then the next-hop address is selected from the default router list.

Neighbor Unreachability Detection (NUD)

Each entry in the Neighbor Cache contains the IP address, the link-layer (MAC) address, and the *reachability status* for that node. There are five possible values for that status, and the state transition rules are as follows:

- INCOMPLETE – cache entry is newly created, and address resolution is in progress. Any transmitted packets are queued. When the address resolution completes, the link-layer address is added into the Neighbor Cache, and the state changes to REACHABLE.

- REACHABLE – any queued packets are immediately sent. Any newly transmitted packets are sent normally. If more than a certain time passes without any traffic to or from the address, the state changes to STALE.

- STALE – the reachability of the node is *UNKNOWN*. The address remains in this state until traffic to that node is generated. At that point, the traffic is queued and the state changes to DELAY.

- DELAY – the address remains in the DELAY state for a short period. The status is still *UNKNOWN*. Once the delay expires, the probe packet is sent, and the state changes to PROBE.

- PROBE – a probe packet has been sent to determine reachability (after the delay), but the result has not yet been obtained. The status is still *UNKNOWN*. When the result is seen, REACHABILITY is confirmed, the state changes to REACHABLE. If a certain amount of time elapses without any response, then the node is considered unreachable, any queued traffic is discarded, and an error is generated to the sender.

Note that there is nothing comparable to Neighbor Unreachability Detection in IPv4. IPv6 NUD improves packet delivery in the presence of failing routers, and over partially failing or partitioned links. It improves delivery to nodes that change their link-layer (MAC) addresses. For example, mobile nodes can move off-link without losing any connectivity due to stale ARP caches. NUD detects dead routers and dead switches that block access to working routers.

Redirect

A router can send a Redirect message to a packet sender, if there is a better first-hop router, or if the destination is an on-link neighbor. In the first case, the Target Address field contains the link-local address of the better first-hop router. In the second case, the Target Address field contains a copy of the Destination Address. The Destination Address field contains the address of the ultimate packet destination. The router uses its knowledge of the larger environment to generate this information. You might think of a Redirect message as saying something like “There is a bridge out down *that* road – try going down *this* road, instead”.

IPv6 Redirects contain the link-layer (MAC) address of the new first hop, which eliminates the need for an additional packet exchange to resolve the IP address. Unlike with IPv4 Redirects, the recipient of an IPv6 Redirect assumes that the new next-hop is on-link. The IPv6 Redirect is useful on non-broadcast and shared media links. On such links, nodes should not check for all prefixes for on-link destinations.

Viewing the Neighbor Cache

To view the Neighbor Cache in Windows 7:

1. Start a Command Prompt (cmd), and enter the following commands in it.
2. Enter the command **netsh –c "interface ipv6".**
3. At the netsh prompt, enter the command **show interface.**
4. In the resulting list, find the interface index for "Local Area Connection" (say it is 11).
5. At the netsh prompt, enter the command **show neighbors 11** (or whatever interface index) .
6. You should see global unicast addresses, link-local addresses, and a lot of multicast addresses.

```
C:\>netsh -c "interface IPv6"
netsh interface IPv6>show interface

Idx     Met         MTU          State                Name
---  ----------  ----------  ------------  ---------------------------
  1          50  4294967295  connected     Loopback Pseudo-Interface 1
 12          50        1280  disconnected  isatap.infoweapons.com
 13          50        1280  connected     Local Area Connection* 11
 11          10        1500  connected     Local Area Connection

netsh interface IPv6>show neighbors 11

Interface 11: Local Area Connection


Internet Address                              Physical Address   Type
--------------------------------------------  -----------------  -----------
2001:df8:5403:2410::fff2                      00-15-17-30-b8-ec  Reachable (Router)
2001:df8:5403:2410::10:11                     00-e0-81-48-62-7a  Stale
fe80::215:17ff:fe30:b8ec                      00-15-17-30-b8-ec  Reachable (Router)
fe80::230:48ff:fe61:d6be                      00-30-48-61-d6-be  Stale
ff02::2                                       33-33-00-00-00-02  Permanent
ff02::c                                       33-33-00-00-00-0c  Permanent
ff02::16                                      33-33-00-00-00-16  Permanent
ff02::1:2                                     33-33-00-01-00-02  Permanent
ff02::1:3                                     33-33-00-01-00-03  Permanent
ff02::1:ff00:69                               33-33-ff-00-00-69  Permanent
ff02::1:ff00:fff2                             33-33-ff-00-ff-f2  Permanent
ff02::1:ff03:186                              33-33-ff-03-01-86  Permanent
ff02::1:ff10:11                               33-33-ff-10-00-11  Permanent
ff02::1:ff10:14                               33-33-ff-10-00-14  Permanent
ff02::1:ff10:26                               33-33-ff-10-00-26  Permanent
ff02::1:ff13:f5                               33-33-ff-13-00-f5  Permanent
ff02::1:ff2b:6589                             33-33-ff-2b-65-89  Permanent
ff02::1:ff30:b8ec                             33-33-ff-30-b8-ec  Permanent
ff02::1:ff3f:58e5                             33-33-ff-3f-58-e5  Permanent
ff02::1:ff61:d6be                             33-33-ff-61-d6-be  Permanent
ff02::1:ff62:62                               33-33-ff-62-00-62  Permanent
ff02::1:ffc6:ed59                             33-33-ff-c6-ed-59  Permanent
ff02::1:ffce:5d59                             33-33-ff-ce-5d-59  Permanent
ff05::1:3                                     33-33-00-01-00-03  Permanent
```

SEcure Network Discovery (SEND)

Note that there are some potentially exploitable vulnerabilities in ND. ARP in TCP/IPv4 has several well known and easily exploited vulnerabilities, used in many hacking attacks. For details of these, search for "ARP Vulnerabilities Black Hat". You should find an excellent PowerPoint presentation that was presented by Mike Beekey at a Black Hat Briefing security conference. It shows exactly how ARP is vulnerable, and how this is exploited by hackers. APR does not exist in IPv6, so its vulnerabilities do not

affect IPv6 networks. However, ND (which replaces ARP) has some new vulnerabilities, that do not affect IPv4 networks.

A secure version of ND is defined in RFC 3971, "SEcure Neighbor Discovery (SEND)", March 2005. This is still a *Proposed Standard*. SEND uses cryptographically generated addresses which are defined in RFC 2972 "Cryptographically Generated Addresses (CGA)", March 2005 (this is also a *Proposed Standard* and has already been updated by RFCs 4581 and 4982). SEND does not depend on IPsec. It is still very much in an experimental status as of the writing of this book.

5.3.3 – Types of IPv6 Packet Transmission

Unicast, anycast, multicast and broadcast have already been covered in section 5.3.2.2, because in IPv6, this is considered to be part of the addressing model.

5.3.3.1 – IPv6 Broadcast

Most things that you would use broadcast for in IPv4, you would use some form of multicast, with a more restricted scope, in IPv6. A multicast transmission to the address ff01::2 would go to the same nodes (all nodes on local link) as an IPv4 broadcast. However, there are other scopes, such as site, organization and global for multicast, that (unlike IPv4 broadcast) will cross routers, but other than "all nodes in local link", multicast to the wider scopes requires that all recipients intentionally add the necessary multicast address to their node.

5.3.3.2 – IPv6 Multicast

The basic multicast address type has been covered, but there is a lot more to a full multicast system, as you saw in section 3.3.3.2 (IPv4 Multicast). For an in-depth discussion of all aspects of IPv6 Multicast, I recommend Chapter 6, "Providing IPv6 Multicast Services" from the book "Deploying IPv6 Networks", by Ciprian Popoviciu, Eric Levy-Abegnoli and Patrick Grossetete, Cisco Press, 2006.

Multicast exists in IPv4, but there are some serious problems with it, which are resolved in IPv6:

- Not all IPv4 routers support multicast. In general it is difficult to deploy except in a "walled garden", such as the customers of a single ISP like Comcast. In IPv6, support for multicast is mandatory – all compliant routers support it, and it works across ISPs, even worldwide.

- The *Internet Group Management Protocol* (IGMP) is not part of TCP/IPv4, and not all IPv4 routers include it. In IPv6, the *Multicast Listener Discovery* protocol (MLD) is standardized, and is actually just a subset of the ICMPv6 messages. Because of this, all IPv6 compliant routers include it.

- Multicast in IPv4 was an afterthought, grafted on long after the original protocol was designed. In IPv6, multicast was designed in from the beginning, and is present in all address scopes. Multicast link-local addresses are used extensively in SLAAC and other places.

For IPTV applications, IPv6 networks will be the first time that really global Internet TV services can be deployed and work reliably. This is as exciting as when Ted Turner first relayed the signal from his small UHF TV station via a satellite. That breakthrough resulted in WTBS, CNN, CNN Headline News, TNT, Cartoon Network, and indirectly, the entire multi-billion dollar satellite / cable television network industry.

There are many other areas in which working, scalable multicast can be used to improve applications. You could build chat, VoIP or even video conferencing clients that could build fully meshed networks, with each new participant subscribing to all existing client's multicast "channels", and all existing clients subscribing to the new participant's multicast "channel". Even if the initial participant left, all remaining participants would still have a fully functional mesh network. This also eliminates the need for any central exchange point (other than perhaps a search or directory facility to help in setting up the conference and allowing participants to locate each other).

The following standards are relevant to multicast in IPv6:

- RFC 2375, "IPv6 Multicast Address Assignments", July 1998 (Informational)
- RFC 2710, "Multicast Listener Discovery (MLD) for IPv6", October 1999 (Standards Track)
- RFC 3306, "Unicast-Prefix-based IPv6 Multicast Addresses", August 2002 (Standards Track)
- **RFC 3307, "Allocation Guidelines for IPv6 Multicast Addresses", August 2002 (Standards Track)**
- RFC 3590, "Source Address Selection for the Multicast Listener Discover (MLD) Protocol", September 2003 (Standards Track)
- **RFC 3810, "Multicast Listener Discovery Version 2 (MLDv2) for IPv6", June 2004 (Standards Track)**
- RFC 3956, "Embedding the Rendezvous Point (RP) Address in an IPv6 Multicast Address", November 2004 (Standards Track)
- RFC 4489, "A Method for Generating Link-Scoped IPv6 Multicast Addresses", April 2006 (Standards Track)
- RFC 4607, "Source-Specific Multicast for IP", August 2006 (Standards Track)

Multicast Listener Discovery Protocol (MLD)

MLD is used by IPv6 routers to discover the presence of multicast listeners (nodes that wish to receive multicast packets), and the specific multicast addresses to which they want to subscribe. MLD (defined in RFC 2710) is commonly referred to as MLDv1. It is the IPv6 equivalent to IPv4's IGMPv2 (defined in RFC 2236). MLDv1 and IGMPv2 multicast protocols are used to set up *Any-Source Multicast* (ASM), which allows multiple sources in a group (*,G) or "channel". This is also known as *traditional multicast.*

MLDv2 extends the definition of MLDv1 by adding support for "source filtering". It includes all of the functionality of MLDv1, so there is no need to deploy both on a given node. This allows a node to indicate interest *only* in packets from specific source addresses (INCLUDE mode), or in packets from all multicast addresses *except* for specific source addresses (EXCLUDE mode). MLDv2 is the IPv6 equivalent of IPv4's IGMPv3. MLDv2 and IGMPv3 multicast protocols are used to set up *Source-Specific Multicast* (SSM), which allows a specific source (S) in a group (G) to deliver packets to all members that join (S,G)

known as a “channel”. This is described in RFC 4604, “Using Internet Group Management Protocol Version 3 (IGMPv3) and Multicast Listener Discovery Protocol Version2 (MLDv2) for Source-Specific Multicast”, and in RFC 4607, “Source-Specific Multicast (SSM) for IP”.

There is another RFC that defines MLD proxying: RFC 4605, “Internet Group Management Protocol (IGMP) / Multicast Listener Discovery (MLD)-Based Multicast Forwarding (“IGMP/MLD Proxying”)”. A *proxy* would exist on a forwarding gateway that links together multiple subnets, and relay messages across that gateway between an MLD querier on *one* subnet and MLD listeners on *a different* subnet.

MLDv1 and MLDv2 are *sub-protocol*s of ICMPv6. All MLDv2 messages are just additional ICMPv6 messages. All IPv6 compliant devices should include support for MLD. MLD messages must be sent with a link-local IPv6 Source Address, a Hop Limit of 1, and an IPv6 Router Alert Option in the Hop-by-hop option extension packet header. When used in Neighbor Discovery protocol’s *Stateless Address Autoconfiguration*, the source address can be the unspecified address (::). IGMP is *not* a sub protocol of ICMPv4. It does not use ICMPv4 messages, but an entirely new protocol. IGMP is not mandatory on all IPv4 routers.

MLD can co-exist with IGMPv3 in a dual-stack network, as MLD (v1 or v2) will only involve IPv6 messages, and IGMP (v1, v2 or v3) will only involve IPv4 messages. However, in general, multicast will work far better on IPv6 than on IPv4.

With MLD, there is a “router role” (performed by at most one router in a subnet) and a “listener role” (performed by any number of listener nodes in that subnet) in the protocol.

For the router role, only one router on a subnet can be *the Querier* at any given time. If there is more than one router on a subnet, there is an election mechanism that selects one of them to be the Querier. Should that router fail at some point, all other routers on that subnet have been listening in and maintaining state, so another election will select one of the surviving routers on that subnet to become the Querier. Only the Querier sends periodic or triggered query messages on its subnet.

There are three types of MLDv2 query messages sent by the Querier to the “all nodes on local link” multicast address (*ff02::1*). They should be sent with a valid IPv6 link-local source address. Any Query message received with the Source Address being the unspecified address (::), or any other address that is not a valid IPv6 link-local address, should be silently discarded.

- General Queries
- Multicast Address Specific Queries
- Multicast Address and Source Specific Queries

There are two types of reports sent by listeners to the Querier, to a special multicast address (*ff02::16*) to which all MLDv2 compliant multicast routers listen. If a single Report message is not large enough to hold all of the state information, multiple Report messages can be sent.

- Current State Report (sent in response to a Query)
- State Change Report (sent unsolicited in response to some change on the listener)

General Queries are sent from the Querier to all listeners on the subnet periodically to learn multicast address listener information, to build and refresh state inside all multicast routers on the subnet. Even though only the Querier sends out periodic queries, all routers listen to the responses, and update their state.

When a listener node gets a General Query message, it responds by sending a *Current State Report*, with its per-interface state information. It is also possible for a listener node to immediately report a state change (such as someone "unsubscribing" to a multicast channel) through an unsolicited *State Change Report*. Current State Reports are sent only once (if one is lost, it will probably be received in response to the next periodic Query). State Change Reports are sent multiple times for robustness (to increase the probability of all routers getting the message).

When the Querier gets a State Change Report from a listener, it sends a *Multicast Address Specific Query* to see if there are still any *other* listeners to that multicast address. If not, the Querier will delete that multicast address from its Multicast Address Listener state table, which stops relaying the corresponding traffic. If there are source specific listeners, the Querier will send a *Multicast Address and Source Specific Query* instead.

There must be a *service interface* (API routines) available, which allows an application to cause a *State Change Report* to be sent to the Querier. A sample API is documented in RFC 3678, "Socket Interface Extensions for Multicast Source Filters", January 2004. The full API includes the ability to JOIN or LEAVE a multicast group ("subscribe to a multicast channel"), and to BLOCK and UNBLOCK specific source addresses, as well as to set and retrieve source filter sets.

For details on the syntax of the various MLDv2 messages, see RFC 3810.

5.3.3.4 – Protocol Independent Multicast (PIM) for IPv6

PIM is a multicast protocol which deals with router-to-router communications. IPv6 PIM is similar to IPv4 PIM, has the same variants (Dense Mode, Sparse Mode, and Bidirectional Mode), and is defined in the same RFCs (in the sections relevant to IPv6). The IPv6 implementation uses Neighbor Discovery protocol, Multicast Listener Discovery protocol, Path MTU Discovery and IPv6 Multicast, rather than the corresponding IPv4 mechanisms. As with TCP, the PIM message checksum factors in the source and destination IP addresses, so the pseudo-header used in the calculation of the checksum (which includes IPv6 addresses) is different from the one used in IPv4. The following items are IP version specific in all variants:

Item	IPv4	IPv6
Source-Specific Multicast	232.0.0.0/8	ff3x:/32
Wildcard Group set	224.0.0.0/3	ff00::/8
ALL-PIM-ROUTERS group	224.0.0.13	ff02::d

PIM for IPv6 does not include routing, but provides multicast forwarding by using static IPv6 routes, or routing tables created by IPv6 unicast routing protocols, such as RIPng, OSPFv3, IS-ISv6 or BGP4+.

PIM Dense Mode is defined in RFC 3973, "Protocol Independent Multicast – Dense Mode (PIM-DM)", January 2005 (for both IPv4 and IPv6). This uses dense multicast routing, which builds shortest-path trees by flooding multicast traffic domain wide, then pruning branches where no receivers are present. It does not scale well.

PIM Sparse Mode is defined in RFC 4601, "Protocol Independent Multicast – Sparse Modem (PIM-SM): Protocol Specification (Revised)", August 2006 defines PIM-SM (for both IPv4 and IPv6). As in IPv4, PIM-SM builds unidirectional shared trees routed at a rendezvous point per group, and can create shortest-path trees per source. It scales fairly well for wide-area use.

Bidirectional PIM is defined in RFC 5015, "Bidirectional Protocol Independent Multicast (BIDIR-PIM)", October 2007 (for both IPv4 and IPv6). It builds shared bi-directional trees. In never builds a shortest-path tree, so there may be longer end-to-end delays, but it scales very well.

There is one new standard specific to IPv6 PIM, RFC 3956 "Embedding the Rendezvous Point (RP) Address in an IPv6 Multicast Address", November 2004. This defines an address allocation policy in which the address of the Rendezvous Point (RP) is encoded in an IPv6 multicast group address. For PIM-SM, this can be seen as a specification of a group-to-RP mapping mechanism. This supports easy deployment of scalable inter-domain multicast and simplifies configuration as well.

Example 1: An ISP manages 2001:db8::/32, and wants an RP for the network and all its customers, on an existing subnet, for example 2001:db8:beef:feed::/64. The group address would be something like ff7x:y40:2001:db8:beef:feed::/96, and the RP address would be 2001:db8:beef:feed::y (y can be any value from 1 to F, but not 0).

Example 2: An organization wants to have its own PIM-SM domain. It should pick multicast addresses such as ff7x:y30:2001:db8:beef::/80. The RP address would be 2001:db8:beef::y (y can be any value from 1 to F, but not 0).

5.3.4 – ICMPv6: Internet Control Message Protocol for IPv6

ICMPv6 is a key protocol in the *Internet Layer* that complements version 6 of the Internet Protocol (IPv6). It was originally defined in RFC 1885 (December 1995) and then enhanced in RFC 2463 (December 1998). It is currently defined in RFC 4443, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", March 2006.

There are many more ICMPv6 messages defined than there are ICMPv4 messages (in fact, Neighbor Discovery and Multicast Listener Discovery protocols are just subsets of the ICMPv6 messages). ICMPv6 messages have a much greater range of functionality than ICMPv4 messages. Even if you block all ICMPv4 messages (common practice by some IPv4 network administrators) normal network operation will usually occur. *This is not true with ICMPv6.* ICMPv6 messages are used in normal operation of IPv6.

There are two classes of ICMPv6 messages:

- error messages, with message type ranging from 0 to 127
- informational messages, with message type ranging from 128 to 255

ICMPv6 Error Messages

1	Destination Unreachable (ICMPv6, RFC 4443)
2	Packet Too Big (ICMPv6, RFC 4443)
3	Time Exceeded (ICMPv6, RFC 4443)
4	Parameter Problem (ICMPv6, RFC 4443)

ICMPv6 Informational Messages

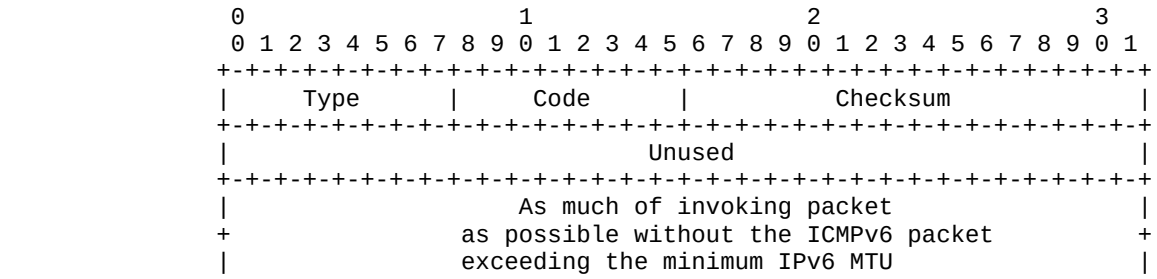
128	Echo Request (ICMPv6, RFC 4443)
129	Echo Reply (ICMPv6, RFC 4443)
130	Multicast Listener Query message (MLDv2, RFC 3810)
131	Multicast Listener Report (MLDv1, RFC 2710)
132	Multicast Listener Done (MLDv1, RFC 2710)
133	Router Solicitation message (ND, RFC 2461)
134	Router Advertisement message (ND, RFC 2461)
135	Neighbor Solicitation message (ND, RFC 2461)
136	Neighbor Advertisement message (ND, RFC 2461)
137	Redirect message (ND, RFC 2461)
138	Router Renumbering (RR, RFC 2894)
139	ICMP Node Information Query (NIQ, RFC 4620)
140	ICMP Node Information Response (NIQ, RFC 4620)
141	Inverse Neighbor Discovery Solicitation Message (IND, RFC 3122)
142	Inverse Neighbor Discovery Advertisement message (IND, RFC 3122)
143	Multicast Listener Report message (MLDv2, RFC 3810)
144	Home Agent Address Discovery Request Message (MIPv6, RFC 3775)
145	Home Agent Address Discovery Reply Message (MIPv6, RFC 3775)
146	Mobile Prefix Solicitation (MIPv6, RFC 3775)
147	Mobile Prefix Advertisement (MIPv6, RFC 3775)
148	Certification Path Solicitation (SEND, RFC 3971)
149	Certification Path Advertisement (SEND, RFC 3971)
151	Multicast Router Advertisement (MRD, RFC 4286)
152	Multicast Router Solicitation (MRD, RFC 4286)
153	Multicast Router Termination (MRD, RFC 4286)
154	FMIPv6 messages (MIPv6, RFC 5568)

IND	Inverse Neighbor Discovery
MIPv6	Mobile IPv6
MLDv1	Multicast Listener Discovery, version 1
MLDv2	Multicast Listener Discovery, version 2
MRD	Multicast Router Discovery
ND	Neighbor Discovery
NIQ	Node Information Query
RR	Router Renumbering
SEND	SEcure Neighbor Discovery

Note that there is no equivalent ICMPv6 message corresponding to the following ICMPv4 messages (or else its function is now contained in another message)

4	Source Quench
5	Redirect
13	Timestamp
14	Timestamp Reply
15	Information Request
16	Information Reply

Destination Unreachable Error



IPv6 Fields:

Destination Address

Copied from the Source Address field of the invoking packet.

ICMPv6 Fields:

Type 1

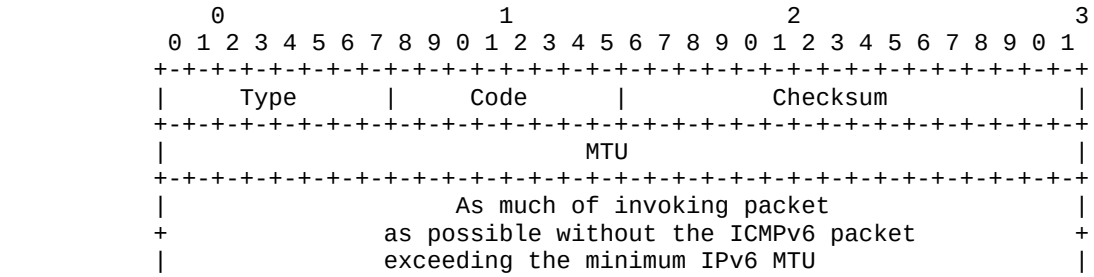
Code
0 - No route to destination
1 - Communication with destination administratively prohibited
2 - Beyond scope of source address
3 - Address unreachable
4 - Port unreachable
5 - Source address failed ingress/egress policy
6 - Reject route to destination

Unused This field is unused for all code values. It must be initialized to zero by the originator and ignored by the receiver.

Description

A Destination Unreachable message SHOULD be generated by a router, or by the IPv6 layer in the originating node, in response to a packet that cannot be delivered to its destination address for reasons other than congestion. (An ICMPv6 message MUST NOT be generated if a packet is dropped due to congestion.)

Packet Too Big Message



IPv6 Fields:

Destination Address

Copied from the Source Address field of the invoking packet.

ICMPv6 Fields:

Type 2

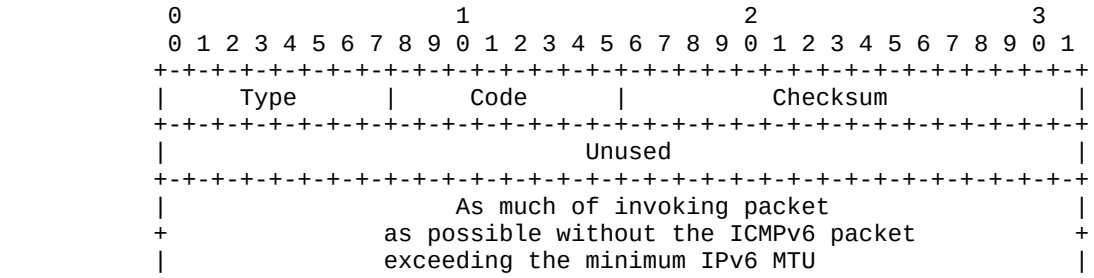
Code Set to 0 (zero) by the originator and ignored by the receiver.

MTU The Maximum Transmission Unit of the next-hop link.

Description

A Packet Too Big MUST be sent by a router in response to a packet that it cannot forward because the packet is larger than the MTU of the outgoing link. The information in this message is used as part of the Path MTU Discovery process.

Time Exceeded Message



IPv6 Fields:

Destination Address
Copied from the Source Address field of the invoking packet.

ICMPv6 Fields:

Type 3

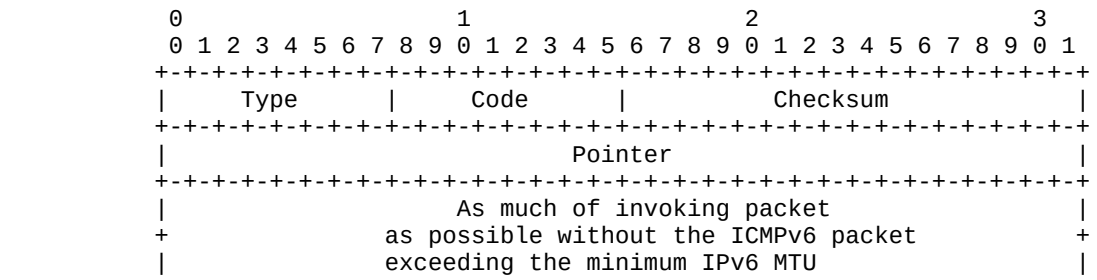
```
Code            0 - Hop limit exceeded in transit
                1 - Fragment reassembly time exceeded

Unused          This field is unused for all code values.
                It must be initialized to zero by the originator
                and ignored by the receiver.
```

Description

If a router receives a packet with a Hop Limit of zero, or if a router decrements a packet's Hop Limit to zero, it MUST discard the packet and originate an ICMPv6 Time Exceeded message with Code 0 to the source of the packet. This indicates either a routing loop or too small an initial Hop Limit value.

Parameter Problem Message



IPv6 Fields:

Destination Address

Copied from the Source Address field of the invoking packet.

ICMPv6 Fields:

Type 4

Code
0 - Erroneous header field encountered
1 - Unrecognized Next Header type encountered
2 - Unrecognized IPv6 option encountered

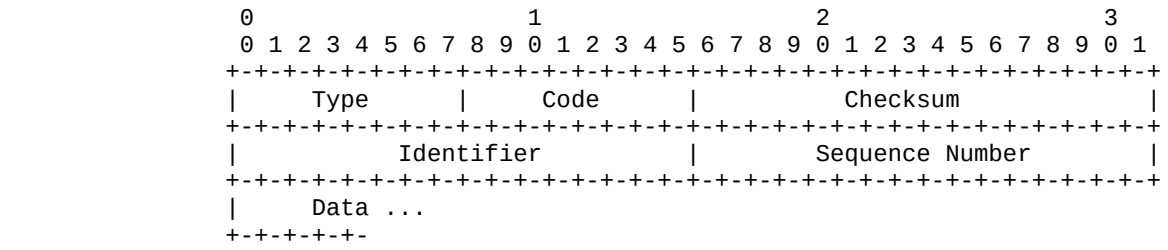
Pointer Identifies the octet offset within the invoking packet where the error was detected.

The pointer will point beyond the end of the ICMPv6 packet if the field in error is beyond what can fit in the maximum size of an ICMPv6 error message.

Description

If an IPv6 node processing a packet finds a problem with a field in the IPv6 header or extension headers such that it cannot complete processing the packet, it MUST discard the packet and SHOULD originate an ICMPv6 Parameter Problem message to the packet's source, indicating the type and location of the problem.

Echo Request Message



IPv6 Fields:

Destination Address

Any legal IPv6 address.

ICMPv6 Fields:

Type 128

Code 0

Identifier An identifier to aid in matching Echo Replies to this Echo Request. May be zero.

Sequence Number

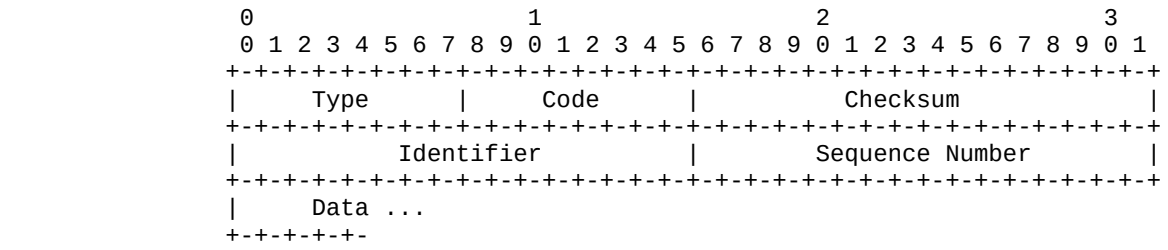
A sequence number to aid in matching Echo Replies to this Echo Request. May be zero.

Data Zero or more octets of arbitrary data.

Description

Every node MUST implement an ICMPv6 Echo responder function that receives Echo Requests and originates corresponding Echo Replies. A node SHOULD also implement an application-layer interface for originating Echo Requests and receiving Echo Replies, for diagnostic purposes.

Echo Reply Message



IPv6 Fields:

Destination Address

Copied from the Source Address field of the invoking

```
                Echo Request packet.

   ICMPv6 Fields:

   Type           129

   Code           0

   Identifier     The identifier from the invoking Echo Request message.

   Sequence Number

                  The sequence number from the invoking Echo Request
                  message.

   Data           The data from the invoking Echo Request message.

   Description

   Every node MUST implement an ICMPv6 Echo responder function that
   receives Echo Requests and originates corresponding Echo Replies.  A
   node SHOULD also implement an application-layer interface for
   originating Echo Requests and receiving Echo Replies, for diagnostic
   purposes.

   The source address of an Echo Reply sent in response to a unicast
   Echo Request message MUST be the same as the destination address of
   that Echo Request message.

   An Echo Reply SHOULD be sent in response to an Echo Request message
   sent to an IPv6 multicast or anycast address.  In this case, the
   source address of the reply MUST be a unicast address belonging to
   the interface on which the Echo Request message was received.

   The data received in the ICMPv6 Echo Request message MUST be returned
   entirely and unmodified in the ICMPv6 Echo Reply message.
```

5.3.5 – IPv6 Routing

TCP/IPv6 has to solve the same problems as TCP/IPv4 in terms of how to get packets from one point to another through a packet switched network. However, the differences in IP address length and addressing model mean that the existing routing protocols for TCP/IPv4 do not work. All of the popular routing protocols have been extended to support IPv6. These include RIPng, EIGRP, IS-ISv6, OSPF for IPv6, and BGP4 with Multiprotocol Extensions (BGP4+).

The following standards are relevant to routing in IPv6:

- **RFC 2080, “RIPng for IPv6”, January 1997 (Standards Track)**
- RFC 2185, “Routing Aspects of IPv6 Transition”, September 1997 (Informational)
- **RFC 2545, “Use of BGP-4 Multiprotocol Extensions for IPv6 Inter-Domain Routing”, March 1999 (Standards Track)**
- RFC 5308, “Routing IPv6 with IS-IS”, October 2008 (Standards Track)
- **RFC 5340, “OSPF for IPv6”, July 2008 (Standards Track)**

RIPng – RIP Next Generation. Defined in RFC 2080, "RIPng for IPv6", January 1997. This IETF standard specifies extensions to the RIP protocol (as defined in RFCs 1058 and 1723), to support IPv6. Like RIP for IPv4, RIPng also uses the Distance Vector algorithm. Unlike RIP for IPv4, RIPng is implemented only in routers. IPv6 itself provides mechanisms for router discover (part of ND). RIPng is a UDP based protocol, using port 521 (compare with port 520 for RIP). It supports 128 bit IPv6 addresses instead of 32 bit IPv4 addresses. It has the same limitations as RIP, such as being useful only in small networks, with less than 15 hops. It does have some of the extensions of RIPv2. When a response is sent to all neighbors, the multicast group *ff02::9* (all-rip-routers) is used. RIPng only routes IPv6. On a dual stack network, you would need both RIP (for IPv4) and RIPng. Since RIPng runs over IPv6, it can use the IPsec Authentication Header (AH) and Encapsulating Security Payload (ESP) mechanisms to ensure integrity and authentication / confidentiality of routing exchanges.

EIGRP – Enhanced Interior Gateway Routing Protocol (proprietary Cisco routing protocol). This already includes extensions to allow it to route IPv4 and/or IPv6 packets. For details, see Cisco documentation.

IS-ISv6 – Extension of IS-IS to support IPv6. Based on two levels, L2 = Backbone, L1 = Stub, L2L1 = interconnected L2 and L1. It runs over CLNS (ConnectionLess Network Service, an OSI network layer protocol, similar to IP). Each IS node still sends out Link State Packets, and sends information via Tag/Length/Values. There are two new TLVs: IPv6 Reachability and IPv6 Interface Address, and a new Network Layer Identifier: IPv6 NLPID (Network Layer Protocol IDentifier). Other than that, IS-ISv6 is pretty much the same as the original IS-IS. It is still suitable mainly for large ISPs.

OSPF for IPv6 – Open Shortest Path First for IPv6 (also known as *OSPFv3*). Defined in RFC 5340, "OSPF for IPv6", July 2008. This is still an interior gateway routing protocol, and is suitable for use within organizations, but not between autonomous systems (BGP4+ is needed for this).

The basic OSPF for IPv4 mechanisms (flooding, Designated Router election, area support, Short Path First calculations) are unchanged. Some changes are required because of new protocol semantics or larger address size. Most fields and packet-size limitations in OSPF for IPv4 have been relaxed, and option handling is more flexible. The protocol processing is now per-link, instead of per-subnet. There is now a *flooding scope* to reflect the scopes of IPv6 addresses. It uses IPv6 link-local addresses. The Addressing Semantics have been removed (with a few exceptions) leaving a mostly network-protocol-independent core. OSPF Router IDs, Area IDs and Link State IDs are still 32 bits, so those can no longer be IP addresses (which in IPv6 are 128 bits).

The new *flooding scope* allows control over how widely to flood information: link-local, area wide or AS wide (the entire routing domain). It is now possible to run multiple instances of the OSPF protocol on a single link (every message now includes an *Instance ID* value). Link-local addresses are used where they are meaningful (for transactions completely within a link), but global scope IPv6 addresses must still be used in some places (e.g. source address for OSPF protocol packets). The AuType and Authentication fields have been removed from OSPF for IPv4, as IPsec AH and ESP are available and superior. As with TCP, the header checksum covers the entire OSPF packet and a prepended IPv6 pseudo header. All support for MOSPF (Multicast OSPF) has been removed.

OSPF for IPv6 runs only over IPv6, and only routes IPv6. On a dual stack network you would need both OSPF for IPv4 (OSPFv2) and OSPF for IPv6 (OSPFv3) deployed, similar to RIP and RIPng. It is possible that a future version of OSPF will support both IPv4 and IPv6 routing.

BGP4 with Multiprotocol Extensions (also known informally as *BGP4+*). Defined in RFC 4760, "Multiprotocol Extensions for BGP-4", January 2007. BGP-4 is currently defined in RFC 4271, "A Border Gateway Protocol 4 (BGP-4)", January 2006. BGP-4 supports only IPv4. The multiprotocol extensions have been around since RFC 2283, February 1998, but have been updated with each new version of BGP-4.

These extensions allow BGP4+ to carry routing information for multiple Network Layer protocols, (e.g IPv6, IPX, L3VPN, etc.). L3VPN is a "Layer 3 Virtual Private Network". BGP4+ is designed to be backward compatible, such that a BGP4+ compliant router can exchange IPv4 routing information with a router that does not support the multiprotocol extensions (basic BGP4).

Currently BGP4+ is the primary protocol used for routing IPv6 packets between Autonomous Systems (very large networks under the control of a single entity, such as ISPs or major corporations). Most IPv6 engineers will never work with it, unless they work for an ISP or a really large company.

One of the issues that ISPs face when supporting IPv6 is to migrate their BGP-4 gateways to BGP4+ gateways. They typically must also upgrade many routers to dual stack. At the ISP level, many routers have hardware acceleration, so this can be expensive. These may involve "forklift" upgrades, where entirely new high end routers must be purchased, and there may be relatively little resale value for legacy IPv4-only equipment (hint to ISPs: migrate to IPv6 now and sell your old gear while is still has SOME value!).

Looking at Local Routing Information

In Windows, you can view all currently known routes with the "route print" command. If you have enabled the IPv6 protocol, and are connected to an IPv6 network, you might see something like the following (the "-6" tells it to print only IPv6 route information):

```
C:\Users\lhughes>route print -6
===========================================================================
Interface List
 11...00 18 8b 78 da 1a ......Broadcom NetXtreme 57xx Gigabit Controller
  1...........................Software Loopback Interface 1
 12...00 00 00 00 00 00 00 e0 Microsoft ISATAP Adapter
 13...00 00 00 00 00 00 00 e0 Teredo Tunneling Pseudo-Interface
===========================================================================

IPv6 Route Table
===========================================================================
Active Routes:
 If Metric Network Destination      Gateway
 11     26 ::/0                     fe80::21b:21ff:fe1e:f4
  1    306 ::1/128                  On-link
 13     58 2001::/32                On-link
 13    306 2001:0:cf2e:3096:30c3:380d:f5fd:fa12/128
                                    On-link
```

```
 11     18 2001:df8:5403:2410::/64  On-link
 11    266 2001:df8:5403:2410:3c79:b2ca:90ce:5d59/128
                                    On-link
 11    266 2001:df8:5403:2410:a446:d5ef:d313:f5/128
                                    On-link
 11    266 fe80::/64               On-link
 13    306 fe80::/64               On-link
 13    306 fe80::30c3:380d:f5fd:fa12/128
                                    On-link
 11    266 fe80::3c79:b2ca:90ce:5d59/128
                                    On-link
  1    306 ff00::/8                On-link
 13    306 ff00::/8                On-link
 11    266 ff00::/8                On-link
===========================================================================
Persistent Routes:
  None
```

5.3.5.1 – Network Address Translation

NAT (Network Address Translation) was introduced to extend the lifetime of the IPv4 address space long enough for its replacement, IPv6, to be defined, refined, and compliant infrastructure products and applications to be developed. TCP/IPv6 is now fully developed and ready for prime time. NAT has served its purpose. It is time to put it out to pasture.

There is a common belief that the practice of hiding nodes behind a single routable IPv4 address ("hide mode NAT") adds security. It really doesn't.

First, anytime you make an outgoing connection, either directly, or via NAT, the connection you make is a two-way path, and the node you connect to can easily attack you right through your packet filtering firewall and network address translation. You should have "defense in depth" and protect your node with a host-based firewall whether or not you are behind a firewall and NAT gateway.

Second, if a hacker manages to breach your firewall by installing a Trojan horse onto any node in your network, they can attack you from that compromised node. Hackers have a term for networks that have a strong perimeter defense, but limited internal defenses. It is "hard crunchy outside, soft chewy inside". Again, host based firewalls on all nodes are a good idea.

Third, if you are using almost any peer-to-peer software, VoIP (e.g. Skype) or IPsec VPN, it probably includes a mechanism called NAT Traversal (e.g. STUN, TURN, SOCKS, etc.). NAT traversal basically bores a hole right through your NAT protection (required for any of the above applications). Anything that includes NAT traversal can easily be used to attack you. Many people think Skype is a productivity tool. Network security people think it is a security vulnerability.

Fourth, any time you open a document from outside (Word Document, Excel spreadsheet, JPEG image, etc.) it may contain malware that infects your node right through firewalls and NAT.

It is better to allow direct connections to your node over IPv6, through various layers of firewalls, including a host based firewall, together with good active anti-malware software, than to have NAT giving you a false sense of security.

On the other hand, NAT causes problems with any connectivity model other than simple client server outgoing connections, such as web browser to web server. This was covered in some detail under NAT in Chapter 3, section 3.3.5.1.

The real kicker is that NAT is the hacker's friend! It is easy for a hacker to hide behind a NAT gateway and do all kinds of mischief, sending of malware, etc. It is quite difficult for the authorities to figure out which of the nodes hidden behind the common address is doing the bad stuff. To do this, the ISP must log EVERY connection, including source address, destination address, timestamp and port. This mounts up to several TERABYTES for each ISP customer over a year, which is not a trivial amount of storage. With a flat address space (as in IPv6), it is far easier to figure out where the attack is coming from.

Because of these issues, *there is no IPv6 to IPv6 NAT* defined in any IETF standard. There is no need for it to extend the IPv6 address space lifetime, it has no other real benefit, it causes many problems, and it is greatly impeding innovation. Other than those minor things, I guess it's OK (sarcasm warning!).

On the other hand, there is a real need for IPv4 to IPv6 (and IPv6 to IPv4) Network Address Translation, and there are about 8 proposed methods in the IETF now. All of them have various problems and tradeoffs (that is the nature of NAT). One of the more promising schemes is NAT64 in combination with DNS64. These will be discussed in more detail in the chapter on migration to TCP/IPv6.

5.4 – TCP: The Transmission Control Protocol

There is very little difference in TCP over IPv4 and TCP over IPv6. The main difference is that more storage must be provided in the implementations to hold the 4X larger addresses (16 bytes versus 4 bytes, for each address). The other aspect involves the TCP header checksum, which uses a *pseudo header* to allow inclusion of the IP addresses in the calculation of the checksum (in addition to the contents of the payload). Of course there is a different pseudo header format for IPv4 and IPv6, given the difference in address size. There are no new RFCs for TCP over IPv6.

There is one new feature for both TCP and UDP over IPv6 called "Jumbograms" . This is defined in RFC 2675, "IPv6 Jumbograms", August 1999. Jumbograms are very large packets, with a payload containing more than 65,535 bytes. The standard *Payload Length* field is only 16 bits, so the maximum payload size is 65,535 bytes. RFC 2675 defines a new Hop-by-hop option that includes a 32 bit payload length field, allowing packet lengths of up to 4.3 billion bytes. Of course, such packets require paths with very large MTUs. The simple 16 bit checksum becomes a less reliable error detection scheme as the payload length increases significantly. Of course, one bit error would require retransmission of an entire packet, so this should be used only on extremely reliable links.

5.4.1 – TCP Packet Header

No changes are required to the TCP packet header, as port numbers are still 16 bits in length. The only differences are in how the header checksum is calculated (using the IPv6 pseudo header), and the availability of Jumbograms. For details on the TCP packet header fields, see section 3.4.1.

5.5 – UDP: The User Datagram Protocol

UDP over IPv6 has basically the same differences with UDP over IPv4 as was described for TCP. For details on the UDP header fields, see section 3.5.

5.6 – DHCPv6 – Dynamic Host Configuration Protocol for TCP/IPv6

Unlike with DNS, it was not possible to add new functionality into DHCPv4 to make a new version for IPv6 (let alone a single server that could handle both IPv4 and IPv6). DHCPv6 is pretty much a new design from the ground up. DHCPv4 was built from an earlier protocol called BOOTP, and contains many now unnecessary features from that. DHCPv6 was cleaned up considerably, and contains none of the things leftover from BOOTP.

DHCPv4 runs over IPv4, and supplies only 32-bit IPv4 information (assigned IPv4 addresses, IPv4 addresses of DNS servers, etc.). DHCPv6 runs only over IPv6, and supplies only 128-bit IPv6 information (assigned IPv6 addresses, IPv6 addresses of DNS servers, etc.). There is no conflict between DHCPv4 and DHCPv6 in terms of functionality or ports used, so it is possible to run both on a single, dual-stack node.

Hosts communicate only with DHCPv6 servers or relay agents on their local link, using link-local addresses (typically *ff02::1:2,* "all DHCPv6 relay agents and servers"). DHCPv6 uses ports UDP 546 and 547 (compare with DHCPv4 which uses UDP ports 67 and 68). As with DHCPv4, relay agents are used to allow hosts to communicate with remote DHCPv6 servers (ones not on the local link). This is still done via UDP, but using a site-scope address (*ff05::1:3* "all DHCPv6 servers, but not relay agents") which is used only by relay agents.

In some simple networks, there is no need for DHCPv6 because of Stateless Address Autoconfiguration. Currently, however, DHCPv6 is the only way for IPv6 capable nodes to automatically learn the IPv6 addresses of DNS servers. This is particularly important for IPv6-only ("pure IPv6") networks, of which there are not many yet. For dual stack networks, there is no conflict between DHCPv4 and DHCPv6, and both can exist even on a single node. In this case, the IPv4 side of a node would get its IPv4 configuration from the DHCPv4 server, and the IPv6 side of a node would get its IPv6 configuration from the DHCPv6 server.

DHCPv6 allows the administrator far better control over distribution of interface identifiers (low 64 bits of each address) than with Stateless Address Autoconfiguration. With SLAAC, interface identifiers can either make use of only a tiny percentage of the possible 2^{64} address space (when using EUI-64 generated interface identifiers), or have interface identifiers scattered randomly all over the possible 2^{64} address space (when using cryptographically generated addresses). Either of these can lead to problems with Network Access Control or Firewall rules. In general, administrators like to cluster IP addresses by department (or other groupings), so that a single firewall or NAC rule can be used for an entire group, by specifying an *address range* (e.g. all addresses that fall between *2001:df8:5403:3000::1000* and *2001:df8:5403:3000::1fff*, inclusive).

IPv6 capable nodes can be informed that there is a DHCPv6 server available via two bits in the Router Advertisement message. The Router Advertisement message and the relevant bits are described in RFC

4861, “Neighbor Discovery for IP version 6 (IPv6)”, September 2007. In the Router Advertisement message there are two bits, M and O (first and second bits of the sixth byte of the Router Advertisement message), with the following semantics:

M – “Managed address configuration” flag. When set it indicates that addresses are available via DHCPv6. If set, then the O flag can be ignored. This enables *stateful DHCPv6*, where both the stateless information (IPv6 addresses of DNS and other servers) *and* global unicast addresses can be obtained from DHCPv6.

O – “Other configuration” flag. When set, it indicates that other configuration information is available via DHCPv6. This includes things such as IPv6 addresses of DNS or other servers. This is called *stateless DHCPv6*, and is used in conjunction with Stateless Address Autoconfiguration (for obtaining global unicast addresses).

If both M and O bits are clear, then SLAAC is the only way to get addresses, and there is no source of IPv6 addresses for any servers, including DNS.

RFCs for DHCPv6

There are a number of RFCs that define DHCPv6. The most important ones are:

- **RFC 3315, “Dynamic Host Configuration Protocol for IPv6 (DHCPv6)”, July 2003 (Standards Track)**
- RFC 3319, “Dynamic Host Configuration Protocol (DHCPv6) Options for Session Initiation Protocol (SIP) Servers”, July 2003
- RFC 3633, “IPv6 Prefix Options for Dynamic Host Configuration Protocol (DHCP) version 6”, December 2003 (Standards Track)
- **RFC 3646, “DNS Configuration options for Dynamic Host Configuration Protocol for IPv6 (DHCPv6)”, December 2003 (Standards Track)**
- **RFC 3736, “Stateless Dynamic Host Configuration Protocol (DHCP) Service for IPv6”, April 2004 (Standards Track)**
- RFC 3898, “Network Information Service (NIS) Configuration Options for Dynamic Host Configuration Protocol for IPv6 (DHCPv6)”, October 2004 (Standards Track)
- RFC 4076, “Renumbering Requirements for Stateless Dynamic Host Configuration Protocol for IPv6 (DHCPv6)”, May 2005 (Informational)
- RFC 4242, “Information Refresh Time Option for Dynamic Host Configuration Protocol for IPv6 (DHCPv6)”, November 2005 (Standards Track)
- **RFC 4477, “Dynamic Host Configuration Protocol (DHCP): IPv4 and IPv6 Dual-Stack Issues”, May 2006 (Informational)**
- RFC 4580, “Dynamic Host Configuration Protocol for IPv6 (DHCPv6) Relay Agent Subscriber-ID Option”, June 2006 (Standards Track)
- RFC 4649, “Dynamic Host Configuration Protocol for IPv6 (DHCPv6) Relay Agent Remote-ID Option”, August 2006 (Standards Track)
- RFC 4704, “The Dynamic Host Configuration Protocol for IPv6 (DHCPv6) Client Fully Qualified Domain Name (FQDN) Option”, October 2006 (Standards Track)
- RFC 5007, “DHCPv6 Leasequery”, September 2007 (Standards Track)

- RFC 5460, “DHCPv6 Bulk Leasequery”, February 2009 (Standards Track)

DHCPv6 has a failover mechanism. Two servers can manage a single pool of addresses for redundancy (in case of failure of one of the servers). This also can be used for load balancing.

All IPv6 hosts have automatically generated *link-local* addresses that can be used to exchange packets with any other node on the local link. DHCPv4 requires some complex hacks to allow hosts to communicate before they get an address. All IPv6 hosts support link-local multicast. All DHCPv6 servers listen to DHCPv6 multicast groups. With DHCPv4, clients have to do a general broadcast to all nodes on the link, which generates significant broadcast traffic on the link and unnecessary traffic handling on all nodes.

With DHCPv6, a single request can configure all interfaces on a node. The server can offer multiple addresses, one for each interface, and each interface can even have different options. With DHCPv4, each interface would require a separate DHCP operation.

Some of the *stateless* information (that is, other than assigned IPv6 addresses for each node), include:

- IPv6 prefix
- Vendor specific options
- Addresses of SIP servers
- Addresses of DNS servers and search options
- NIS configuration
- SNTP servers

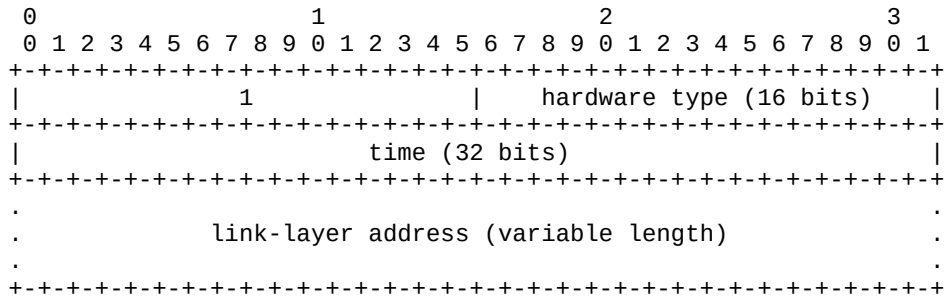
There are several implementations of DHCPv6 already on the market. Windows Server 2008 contains a very complete implementation, in addition to its DHCPv4 server. You can view the IPv6 Ready Phase 2 product list for other options, including my own company’s *SolidDNS* appliance. Some implementations only support *stateless* mode, which means they can supply stateless information (like DNS addresses) but not actually allocate addresses. Be sure the DHCPv6 server you select includes full support for stateful mode as well (where it can supply addresses to each node, in addition to stateless information). You should also be sure that the gateway router or firewall you select has the ability to inform nodes that DHCPv6 servers are available on the subnet.

Address Reservations with DHCPv6

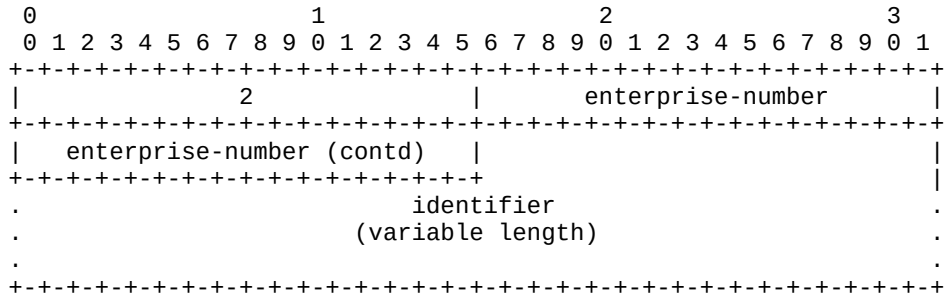
In the case of DHCPv4, it is possible to make an *address reservation*, linked to the MAC address of a node. Anytime the node with a MAC address for which an address reservation has been made asks DHCPv4 for an address, it will get the specific address that was reserved for that MAC address. In the case of DHCPv6, the same concept applies, except that you use two identifiers called the IAID (Interface Association ID) and DUID (DHCP Unique IDentifier).

A DUID consists of a two-byte type code represented in network byte order, followed by a variable number of bytes that make up the actual identifier. A DUID can be no more than 128 bytes long (not including the type code). The following types are currently defined:

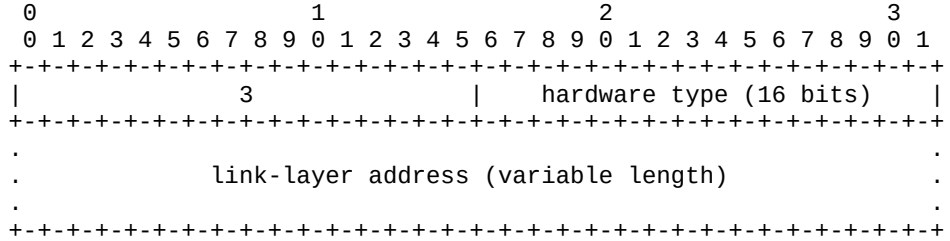
Link-Layer address plus Time **(DUID-LLT)** – This type is recommended for all general purpose computing devices, such as desktop computers, printers, routers, etc. They must contain some form of writable non-volatile storage. Note that the device should configure the time on the node before this DUID is generated, if possible. The only purpose of the timestamp is to lower the chance of an identifier conflict. The link-layer address is typically the MAC address for Ethernet media. The DUID is defined as follows:



Vendor-Assigned Based on Enterprise Number **(DUID-EN)** – This type is assigned to the device by the vendor. This type of DUID is for devices that have some form of non-volatile storage (e.g. EEPROM). The enterprise number is the IANA 32-bit assigned number for the vendor. The identifier can be anything the vendor chooses, but must be unique within that vendor for each device.



Link-Layer Address **(DUID-LL)** – This type is just like the DUID-LLT, without the timestamp. It is recommended for permanently connected devices that have a link-layer address, but no non-volatile, writeable stable storage.



Viewing your Node's DUID

In Windows 7, using a command prompt, type the command **ipconfig /all**. In the section related to the interface you are interested in, look for the field *DHCPv6 Client DUID*. Note that this is a DUID-LLT (Type code 00-01). The next six hex digit pairs (00-01-12-D6-97-E5) are the timestamp. The last six hex digit pairs (00-18-8B-78-DA-1A) are the same as the *Physical Address* (MAC address).

```
Ethernet adapter Local Area Connection:

   Connection-specific DNS Suffix  . : infoweapons.com
   Description . . . . . . . . . . . : Broadcom NetXtreme 57xx Gigabit Controller
   Physical Address. . . . . . . . . : 00-18-8B-78-DA-1A
   DHCP Enabled. . . . . . . . . . . : Yes
   Autoconfiguration Enabled . . . . : Yes
   IPv6 Address. . . . . . . . . . . : 2001:df8:5403:2410:3c79:b2ca:90ce:5d59(Pr
eferred)
   Temporary IPv6 Address. . . . . . : 2001:df8:5403:2410:882:cf5c:e810:363d(Pre
ferred)
   Link-local IPv6 Address . . . . . : fe80::3c79:b2ca:90ce:5d59%11(Preferred)
   IPv4 Address. . . . . . . . . . . : 10.2.5.237(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.240.0.0
   Lease Obtained. . . . . . . . . . : Friday, March 12, 2010 10:52:52 AM
   Lease Expires . . . . . . . . . . : Friday, March 12, 2010 4:11:44 PM
   Default Gateway . . . . . . . . . : fe80::21b:21ff:fe1e:f4%11
                                       10.0.0.10
   DHCP Server . . . . . . . . . . . : 10.1.0.14
   DHCPv6 IAID . . . . . . . . . . . : 234887307
   DHCPv6 Client DUID. . . . . . . . : 00-01-00-01-12-D6-97-E5-00-18-8B-78-DA-1A

   DNS Servers . . . . . . . . . . . : 2001:df8:5403:2400::14
                                       2001:df8:5403:2400::13
                                       10.1.0.14
                                       10.1.0.13
   NetBIOS over Tcpip. . . . . . . . : Enabled
   Connection-specific DNS Suffix Search List :
                                       cebu.infoweapons.com
```

You can also see a *DHCPv6 IAID* value (in this case 234887307). This identifies a particular *Identity Association* which allows a server and a client to identify, group, and manage a set of related IPv6 addresses. Each IA consists of an IAID, one or more IPv6 addresses, and the time T1 and T2 for that IA. Each IA is associated with exactly one interface. For further details, see RFC 3315, section 11.

DHCPv6 Ports and Messages

Clients and servers exchange DHCPv6 messages using UDP over IPv6. The client uses a link-local address, or addresses obtained via other mechanisms as the source address for transmitted and receiving DHCPv6 messages. Servers receive messages from clients using a reserved link-scoped multicast address, so that clients don't need to be configured with the addresses of DHCPv6 servers. To allow hosts to communicate with servers on other links, DHCPv6 relay agents are used. Clients listen for DHCPv6 messages on UDP port 546. Servers and relay agents listen for DHCPv6 messages on UDP port 547.

The link-scoped multicast address used by a client to communicate with an on-link relay agent or server is *ff02::1:2*. All DHCPv6 servers and relay agents are members of this multicast group.

The site-scoped multicast address used by a relay agent to communicate with servers is *ff05::1:3*, if it wants to send a message to all DHCPv6 servers, or does not know the unicast address of the servers. All DHCPv6 servers in given site are members of this multicast group.

There are a number of DHCPv6 messages:

The *SOLICIT* message (1), is sent (multicast) by a client to locate servers.

The *ADVERTISE* message (2), is sent (multicast) by a server to indicate that it is available to provide DHCPv6 service, in response to a Solicit message from a client.

The *REQUEST* message (3) is sent (unicast) by a client to request configuration parameters, including IP addresses, from a specific server.

The *CONFIRM* message (4) is sent (multicast) by a client to any available server to determine whether the addresses it was assigned are still appropriate on the link to which the client is connected.

The *RENEW* message (5) is sent (unicast) by a client to the server that originally provided the client's address and configuration parameters, to extend the lifetime on the addresses assigned to the client and update other configuration parameters.

The *REBIND* message (6) is sent (multicast) by a client to any available server to extend the lifetimes on the addresses assigned to the client and to update other configuration parameters. This message is sent after a client receives no response to a RENEW message.

The *REPLY* message (7) is sent (unicast) by a server to a client in response to a SOLICIT, REQUEST, RENEW or REBIND message received from a client. A server sends a REPLY message containing configuration parameters in response to an INFORMATION-REQUEST message. It sends a REPLY message in response to a CONFIRM message confirming or denying that the addresses assigned to the client are appropriate on the link to which the client is connected. A server sends a REPLY message to acknowledge receipt or a RELEASE or DECLINE message.

The *RELEASE* message (8) is sent (unicast) by a client to the server that assigned addresses to the client to indicate that the client will no longer use one or more of the assigned addresses.

The *DECLINE* message (9) is sent (unicast) to a server to indicate that the client has determined that one or more addresses assigned by the server are already in use on the link to which the client is connected.

The *RECONFIGURE* message (10) is sent (unicast) by a server to a client to inform the client that the server has new or updated configuration parameters, and that the client should initiate a RENEW/REPLY or INFORMATION-REQUEST/REPLY transaction with the server in order to obtain the updated information.

The *INFORMATION-REQUEST* message (11) is sent (unicast) by a client to a server to request configuration parameters, without the assignment of any IP addresses to the client.

The *RELAY-FORW* message (12) is sent (multicast) by a relay agent to forward messages to servers, either directly or through another relay agent. The received message, either a client message or a RELAY-FORW message from another relay agent, is encapsulated in an option in the RELAY-FORW message.

The *RELAY-REPL* message (13) is sent (unicast) by the server to a relay agent containing a message that the relay agent should then deliver to a client. The RELAY-REPL message may be relayed by other relay agents for delivery to the destination relay agent. The server encapsulates the client message as an option in the RELAY-REPL message, which the relay agent extracts and then relays to the next relay agent or directly to the client.

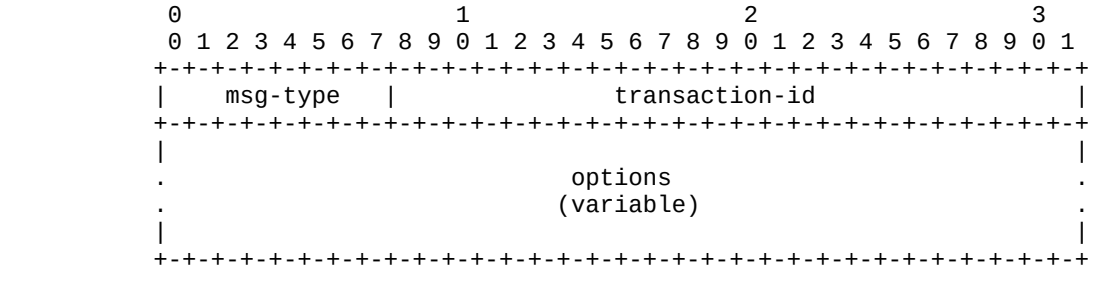
DHCPv6 Status Codes

The following codes are used to communicate the success or failure of operations requested in messages from clients and servers, and additional information about the specific cause in the event of a failure to perform the operation.

```
Name         Code Description
----------   ---- -----------
Success         0 Success.
UnspecFail      1 Failure, reason unspecified; this
                  status code is sent by either a client
                  or a server to indicate a failure
                  not explicitly specified in this
                  document.
NoAddrsAvail    2 Server has no addresses available to assign to
                  the IA(s).
NoBinding       3 Client record (binding) unavailable.
NotOnLink       4 The prefix for the address is not appropriate for
                  the link to which the client is attached.
UseMulticast    5 Sent by a server to a client to force the
                  client to send messages to the server.
                  using the All_DHCP_Relay_Agents_and_Servers
                  address.
```

DHCPv6 Message Syntax

All messages sent between clients and servers share the following syntax:




```
msg-type             Identifies the DHCP message type
```

```
transaction-id       The transaction ID for this message exchange.

options              Options carried in this message.
```

5.6.1 – The DHCPv6 Protocol

DHCPv6 works in somewhat the same was as DHCPv4, except that different messages are used, and communication between client and server take place over using link-local scoped multicast and unicast addresses.

When it first comes up, before any DHCPv6 operation, an IPv6 capable client node obtains a link-local unicast address through ND (and possibly a global unicast address as well, using information from a Router Advertisement message). If a Router Advertisement message is seen, then the client can check the M & O bits in it to determine if there is Stateful DHCPv6, Stateless DHCPv6, or no DHCPv6 available. If no Router Advertisement is available, a client can still attempt DHCPv6 server discovery, as follows.

The client sends a *SOLICIT* message to multicast group *ff02::1:2*. This address specifies *all DHCPv6 servers or relay agents on the local-link*. The included options are:

ClientID
Option Request Option (IA-NA, DNS-Servers, Domain-List)

One or more DHCPv6 servers on the link (or servers on remote links, via DHCPv6 relay agents) will reply with an *ADVERTISE* message to the client that sent the *SOLICIT* message (via unicast). The included options are:

ServerID, ClientID
DNS-Servers, IA-NA (IAID, IAPREFIX).

The client will select one responding DHCPv6 server and send a *REQUEST* message to it (via unicast). This will actually ask for an address lease. The included options are:

ServerID, ClientID
Option Request Option (IA-NA, DNS-Servers, Domain-List)

The selected server will send a *REPLY* message to the client that sent the *REQUEST* message (via unicast). This will confirm the address lease. The included options are:

ServerID, ClientID
DNS-Servers: 2001:xxx:yyy:zzz::a, 2001:xxx:yyy:zzz::b
IA-NA: IAID: 1,
IAPREFIX: Preferred lifetime: nnnnnn,
Valid lifetime: nnnnnn,
Prefix: 2001:xxx:yyy:zzz::c/64

At some later point in time, when the lease is about to expire, the client will send a *RENEW* message to the selected DHCPv6 server. The server will extend the lease and respond with a *REPLY* message.

For Further Information on DHCPv6

For details on how clients send and respond to DHCPv6 messages see RFC 3315, section 17.

For details on DHCP Client-initiated Configuration Exchanges see RFC 3315, section 18.

For details on DHCP Server-initiated Configuration Exchanges see RFC 3315, section 19.

For details on Relay Agent behavior see RFC 3315, section 20.

For details on the optional authentication mechanism, for use of DHCPv6 in unsecured environments, such as wireless networks see section RFC 3315, section 21.

For available DHCPv6 message options and their syntax see RFC 3315, section 22.

Stateless DHCPv6 assumes that assigned IPv6 addresses are obtained some other way, such as Stateless Address Autoconfiguration, and that only stateless information (IPv6 addresses of DNS servers, SIP servers, etc.) will be obtained from DHCPv6. RFC 3736, "Stateless Dynamic Host Configuration Protocol (DHCP) Server for IPv6", April 2004, defines the subset of messages and options from the full (stateful) DHCPv6 functionality that are required to provide stateless DHCPv6 service.

For details on publishing the address of SIP servers with DHCPv6, see RFC 3633, "IPv6 Prefix Options for Dynamic Host Configuration Protocol (DHCP) version 6", December 2003.

For details on publishing the address of DNS servers with DHCPv6, see RFC 3646, "DNS Configuration options for Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", December 2003.

For details on publishing the address of NIS (Network Information Service) servers with DHCPv6, see RFC 3898, "Network Information Service (NIS) Configuration Options for Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", October 2004.

For details on publishing the address of SNTP (Simple Network Time Protocol) servers with DHCPv6, see RFC 4075, Simple Network Time Protocol (SNTP) Configuration Option for DHCPv6", May 2005.

5.6.2 – Useful Commands Related to DHCPv6

In Windows 7, there are some commands available in a command prompt box related to DHCPv6:

ipconfig /release6	release assigned IPv6 address (es), deconfigure network
ipconfig /renew6	do a new configuration request for IPv6
ipconfig /all	view all network configuration settings (IPv4 and IPv6)

This is an example of the output from "ipconfig /all":

```
...
Ethernet adapter Local Area Connection:

   Connection-specific DNS Suffix  . : hughesnet.local
   Description . . . . . . . . . . . : Realtek PCIe GBE Family Controller
   Physical Address. . . . . . . . . : 00-22-15-24-32-9C
   DHCP Enabled. . . . . . . . . . . : Yes
   Autoconfiguration Enabled . . . . : Yes
   IPv6 Address. . . . . . . . . . . : 2001:df8:5403:3000::2:1(Preferred)
   Lease Obtained. . . . . . . . . . : Friday, March 12, 2010 9:43:06 PM
   Lease Expires . . . . . . . . . . : Wednesday, March 24, 2010 9:43:09 PM
   IPv6 Address. . . . . . . . . . . :
                                       2001:df8:5403:3000:b5ea:976d:679f:30f5(Preferred)
   Temporary IPv6 Address. . . . . . :
                                       2001:df8:5403:3000:218a:4956:7d8c:7c2c(Preferred)
   Link-local IPv6 Address . . . . . : fe80::b5ea:976d:679f:30f5%11(Preferred)
   IPv4 Address. . . . . . . . . . . : 172.20.2.1(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.0.0
   Lease Obtained. . . . . . . . . . : Friday, March 12, 2010 9:42:57 PM
   Lease Expires . . . . . . . . . . : Thursday, March 18, 2010 9:43:00 PM
   Default Gateway . . . . . . . . . : fe80::21b:21ff:fe1d:c159%11
                                       172.20.0.1
   DHCP Server . . . . . . . . . . . : 172.20.0.11
   DHCPv6 IAID . . . . . . . . . . . : 218112533
   DHCPv6 Client DUID. . . . . . . . : 00-01-00-01-11-99-BD-28-00-22-15-24-32-9C
   DNS Servers . . . . . . . . . . . : 2001:df8:5403:3000::c
                                       2001:df8:5403:3000::b
                                       172.20.0.11
                                       172.20.0.12
   NetBIOS over Tcpip. . . . . . . . : Enabled
   Connection-specific DNS Suffix Search List : hughesnet.local
...
```

In the above, notice the following:

- The MAC address ("Physical Address") of the interface is *00-22-15-23-32-9C*.

- A 64-bit interface identifier (*b5ea:976d:679f:30f5*) was created, which is a cryptographically generated value (not from EUI-64). A link-local unicast address was generated from this (by prepending *fe80::/64*). The link-local address of the default gateway (*fe80::21b:21ff:fe1d:c159*) was then obtained using ND Router Discovery.

- A Router Advertisement message supplied the subnet prefix (*2001:df8:5403:3000::/64*), so the node used it to create two global unicast addresses, one of which (*2001:df8:5403:3000:b5ea:976d:679f:30f5*) used the 64-bit random interface identifier from ND, the other (*2001:df8:5403:3000:218a:4956:7d8c:7c2c*) used yet another random interface identifier.

- *Obtain an IP address automatically*, and *Obtain DNS server address automatically* were selected in the IPv4 GUI configuration ("DHCP Enabled"), and a working DHCPv4 server was found ("Autoconfiguration Enabled"). So, an IPv4 address (172.20.2.1), the subnet mask (255.255.0.0), the default gateway (172.20.0.1) and the IPv4 addresses of two DNS servers were obtained from the DHCPv4 server. The lease for this was obtained on 3/12/2010, 9:42pm, and will expire on

3/18/2010, at 9:43pm. The MAC address (00-22-15-23-32-9C) was used to make a DHCPv4 reservation for this node, so this node will always get that IPv4 address.

- *Obtain an IPv6 address automatically*, and *Obtain DNS server address automatically* were selected in the IPv6 GUI configuration, and both the M and O bits were set in the Router Advertisement message (Stateful and Stateless DHCPv6 available), so another global unicast IPv6 address (*2001:df8:5403:3000::2:1*) was obtained from DHCPv6, plus the IPv6 addresses of two DNS servers. The lease for this was obtained on 3/12/2010, at 9:43pm and will expire on 3/24/2010 at 9:43pm.

- The DUID of the node is *00-01-00-01-11-99-BD-28-00-22-15-24-32-9C*. The first two hex digit pairs contain 00-01. That means this is a type 1 DUID (DUID-LLT), "Link-Layer plus Timestamp". The next six hex digit pairs (*00-01-11-99-BD-28*) are the timestamp, and the last six digit pairs (*00-22-15-23-32-9C*) contain the interface MAC address. This DUID, along with the IAID (*218112533*) was used to make a DHCPv6 address reservation for this node. So, this node will always get that IPv6 address.

5.7 – TCP/IPv6 Network Configuration

Let's assume our LAN has the following configuration:

Network Prefix:	2001:df8:5403:3000::/64
Default Gateway:	2001:df8:5403:3000::1
DHCPv6 Server Address:	2001:df8:5403:3000::11
DNS Server Addresses:	2001:df8:5403:3000::11, 2001:df8:5403:3000::12
Domain Name:	redwar.org

Furthermore, assume the DHCPv6 server is correctly configured with this information, and is managing the address range 2001:df8:5403:3100::1000 to 2001:df8:5403:3100::1fff (and that some leases have already been granted).

Any node connected to a network with TCP/IPv6 (that will access IPv6 nodes on the Internet) must have certain items configured, including:

- IPv6 link-local node address (obtained automatically)
- All nodes on local-link multicast address (*ff01::1*), there by default
- IPv6 global unicast address
- IPv6 address of default gateway (link-local address of gateway obtained automatically)
- IPv6 addresses of DNS servers (manually configured or from DHCPv6)
- Nodename
- DNS domain name

5.7.1 – Manual Network Configuration for IPv6-Only

It is possible perform TCP/IPv6 configuration manually, either by editing ASCII configuration files, as in FreeBSD or Linux; or via GUI configuration tools, as in Windows. If you have understood the material in this chapter, it should be fairly easy to configure your node(s). In most cases, if you have ISP service, the ISP will give you all the information necessary to configure your node(s). In the coverage of Dual Stack networks we will show configuration of *both* IPv4 and IPv6 on a single node.

Auto Network Configuration Using Stateless Address Autoconfiguration

It is easy for a FreeBSD node to be automatically configured using Stateless Address Autoconfiguration. Note that the global unicast address will be created with the EUI-64 algorithm from your MAC address.

Let's configure a FreeBSD 7.2 node automatically with SLAAC. Assign it the following configuration:

FreeBSD interface name	vr0
nodename	us1.redwar.org
node IP address	(whatever SLAAC comes up with)
default gateway	(whatever SLAAC comes up with)
DNS domain name	redwar.org
DNS Server 1	2001:df8:5403:3000::11
DNS Server 2	2001:df8:5403:3000::12

You need to edit the following files (you will need root privilege to do this):

/etc/rc.conf

```
...
hostname="us1.redwar.org"
IPv6_enable="YES"
...
```

/etc/resolv.conf

```
domain      redwar.org
nameserver  2001:df8:5403:3000::11
nameserver  2001:df8:5403:3000::12
```

If you make these changes, then reboot, you can check the configuration as shown:

```
$ ifconfig vr0
vr0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500
        options=2808<VLAN_MTU,WOL_UCAST,WOL_MAGIC>
        ether 00:15:f2:2e:b4:1c
        inet6 2001:df8:5403:3000::215:f2ff:fe2e:b41c prefixlen 64
        inet6 fe80::215:f2ff:fe2e:b41c%vr0 prefixlen 64 scopeid 0x1
        media: Ethernet autoselect (100baseTX <full-duplex>)
        status: active
$ uname –n
us1.redwar.org
```

```
$ nslookup
> server

> exit

$ netstat –finet6 -rn
```

Auto Network Configuration Using Manually Specified (Static) IPv6 Address

Let's configure a FreeBSD 7.2 node manually with a static node address. Assign it the following configuration:

FreeBSD interface name	vr0
nodename	us1.redwar.org
node IP address	2001:df8:5403:3000::13
default gateway	2001:df8:5403::1
DNS domain name	redwar.org
DNS Server 1	2001:df8:5403:3000::11
DNS Server 2	2001:df8:5403:3000::12

You need to edit the following files (you will need root privilege to do this):

/etc/rc.conf

```
...
hostname="us1.redwar.org"
IPv6_enable="YES"
IPv6_ifconfig_vr0="2001:df8:5403:3000::13 prefixlen 64"
IPv6_defaultrouter="2001:df8:5403:3000::1"
...
```

/etc/resolv.conf

```
domain      redwar.org
nameserver  2001:df8:5403:3000::11
nameserver  2001:df8:5403:3000::12
```

If you make these changes, then reboot, you can check the configuration as shown:

```
$ ifconfig vr0
vr0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500
        options=2808<VLAN_MTU,WOL_UCAST,WOL_MAGIC>
        ether 00:15:f2:2e:b4:1c
        inet6 2001:df8:5403:3000::13 prefixlen 64
        inet6 fe80::215:f2ff:fe2e:b41c%vr0 prefixlen 64 scopeid 0x1
        media: Ethernet autoselect (100baseTX <full-duplex>)
        status: active
$ uname –n
us1.hughesnet.local
$ nslookup
> server
```

```
> exit

$ netstat –finet6 -rn
```

Note: if you specify a static IPv6 Address in FreeBSD 7.x (“IPv6-config_vr0=”…”), the node will not obtain a link-local default gateway address automatically. Therefore in this case it is essential that you also manually specify a default gateway address (which can be global unicast or link-local), using the “IPv6_defaultrouter=…” option in /etc/rc.conf. If no default gateway is defined, communication with other on-link nodes will work OK, but communication with off-link nodes will fail.

This is different from the behavior of Windows 7 and Linux, where the addition of a manually configured global unicast address does not stop the node from obtaining the link-local default gateway automatically.

Chapter 6 – IPsec and Mobile IP

This chapter covers two advanced protocols for TCP/IP, Mobile IP and IPsec. Both of these protocols are available for IPv4 and for IPv6 based networks, but in both cases, there are significant issues with NAT in existing IPv4 networks that severely complicate implementation and deployment. Of course, these issues are not a problem in IPv6 based networks. There are other aspects of IPv6 (support for multicast and link-local addresses) that make both protocols work better on IPv6 than on IPv4. Both of these protocols are quite powerful and add strong incentives for organizations to begin supporting IPv6 sooner rather than later.

6.1 – Internet Protocol Layer Security (IPsec)

The official name for this technology (as used by the IETF) is *The Security Architecture for the Internet Protocol*. Since it takes place in the Internet Layer, protocols at the Transport and Application Layers need not even be aware of it, and do not need to be modified to use it. In effect, you build secure tunnels at the IP layer that the higher layer protocols go through unmodified. Compare with SSL/TLS where the application layer is heavily impacted, both in the design and implementation stage (many changes to source code and application design), and in deployment stage (obtaining and installing server digital certificates into the server application). You could say that applications get a "free ride" over IPsec, gaining security features (without having to include any support for security themselves) simply by running over Internet Layer links secured with IPsec.

By *security* we are here referring to three specific aspects of security:

- *privacy* (keeping others from being able to view the content of your transmissions), accomplished using the Encapsulating Security Payload (ESP) feature
- *authentication* (knowing for sure who the packets came from), accomplished with the Authentication Header (AH) feature
- *message integrity* (knowing if someone has made any changes to the data field or certain of the header fields, including the source and destination addresses), also accomplished with the Authentication Header (AH) feature

The AH and ESP features are mutually independent. You can make use of neither, either or both, depending on your requirements. If you need only authentication and message integrity, you can use only AH. If you need only privacy, you can use only ESP. If you require both, you can use both AH and ESP (there is no conflict between them).

Note however, that since AH protects the source and destination addresses in the IP header and the source and destination ports in the TCP or UDP header, any changes at all to these fields will be detected as tampering (to AH there is no way to distinguish malicious tampering from changes your own network makes to these fields). This means AH will report the changes to IP addresses and ports in the header by NAT to be an attack. It is possible to combine NAT traversal with IPsec (as discussed in RFC 3715, "IPsec-Network Address Translation (NAT) Compatibility Requirements"), but this greatly

complicates the creation and deployment of IPsec applications, and introduces new security issues which may outweigh the benefits of using IPsec in the first place. IPsec is not very suitable for use on existing IPv4 networks since NAT is so widely deployed. Many network professionals have gotten a bad impression of it, but this is because of NAT in IPv4 networks, not any shortcoming of IPsec. There is no other technology for building VPNs supported by the IETF. IPsec works *great* in IPv6 networks. This is because of better support in IPv6 headers to some extent, but primarily because there is no NAT.

Any security features at this level (which must be performed once per packet) must be very lightweight (they cannot use mechanisms that require a lot of CPU power). This rules out the use of asymmetric key cryptography (as used in digital signatures and digital envelopes), at least on a per packet basis. Use of asymmetric key cryptography would cause severe degradation of network throughput. Fortunately, there is a lightweight alternative to digital signatures, which is *Hash-based Message Authentication Codes* (essentially a key driven message digest), which is used in AH. Encryption is handled using only symmetric key encryption and decryption with the same symmetric key for many packets. The key can be manually distributed (*shared secret deployment*) or securely distributed via the Internet Key Exchange (IKE) protocol, which *does* use asymmetric key cryptography (but IKE is used infrequently – the exchanged key is used to encrypt or decrypt a large number of packets before another key is exchanged). Even with these lightweight algorithms, there can still be an impact on network throughput (especially on systems with lower performance CPUs). Network Interface Cards (NICs) are available that include hardware acceleration of the IPsec algorithms (HMAC generation and checking, as well as symmetric key encryption/decryption). These allow wire speed network throughput even on systems with low performance CPUs.

6.1.1 – Relevant Standards for IPsec

The following standards are relevant to IPsec and IKE (also see Appendix A on Cryptography and PKI):

- **RFC 2407, "The Internet IP Security Domain of Interpretation for ISAKMP", November 1998 (Standards Track, *Obsoleted by RFC 4306*)**
- **RFC 2408, "Internet Security Association and Key Management Protocol (ISAKMP)", November 1998 (Standards Track, *Obsoleted by RFC 4306*)**
- **RFC 2409, "The Internet Key Exchange (IKE)", November 1998 (Standards Track, *Obsoleted by RFC 4306*)**
- RFC 2410, "The NULL Encryption Algorithm and Its Use With IPsec", November 1998 (Standards Track)
- RFC 2411, "IP Security Document Roadmap", November 1998 (Informational)
- RFC 2412, "The Oakley Key Determination Protocol", November 1998 (Informational)
- **RFC 2709, "Security Model with Tunnel-mode IPsec for NAT Domains", October 1999 (Informational)**
- **RFC 3193, "Securing L2TP using IPsec", November 2001 (Standards Track)**
- RFC 3554, "On the Use of Stream Control Transmission Protocol (SCTP) with IPsec", July 2003 (Standards Track)
- RFC 3456, "Dynamic Host Configuration Protocol (DHCPv4) Configuration of IPsec Tunnel Mode", January 2003 (Standards Track)
- RFC 3457, "Requirements for IPsec Remote Access Scenarios", January 2003 (Informational)

- RFC 3554, “On the Use of Stream Control Transmission Protocol (SCTP) with IPsec”, July 2003 (Standards Track)
- **RFC 3566, “The AES-XCBC-MAC-96 Algorithm and Its Use with IPsec”, September 2003 (Standards Track)**
- **RFC 3585, “IPsec Configuration Policy Information Model”, August 2003 (Standards Track)**
- **RFC 3602, “The AES-CBC Cipher Algorithm and Its Use with IPsec”, September 2003 (Standards Track)**
- **RFC 3715, “IPsec-Network Address Translation (NAT) Compatibility Requirements”, March 2004 (Informational)**
- RFC 3776, “Using IPsec to Protect IPv6 Signaling Between Mobile Nodes and Home Agents”, June 2004 (Standards Track)
- RFC 3884, “Use of IPsec Transport Mode for Dynamic Routing”, September 2004 (Informational)
- **RFC 3947, “Negotiation of NAT-Traversal in the IKE”, January 2005 (Standards Track)**
- **RFC 3948, “UDP Encapsulation of IPsec ESP Packets”, January 2005 (Standards Track)**
- RFC 4025, “A Method for Storing IPsec Keying Material in DNS”, March 2005 (Standards Track)
- RFC 4106, “The Use of Galois/Counter Mode (GCM) in IPsec Encapsulating Security Payload (ESP)”, June 2005 (Standards Track)
- **RFC 4109, “Algorithms for Internet Key Exchange version 1 (IKEv1)”, May 2005 (Standards Track)**
- RFC 4196, “The SEED Cipher Algorithm and Its use with IPsec”, January 2006 (Standards Track)
- **RFC 4301, “Security Architecture for the Internet Protocol”, December 2005 (Standards Track)**
- **RFC 4302, “IP Authentication Header”, December 2005 (Standards Track)**
- **RFC 4303, “IP Encapsulating Security Payload (ESP)”, December 2005 (Standards Track)**
- RFC 4304, “Extended Sequence Number (ESN) Addendum to IPsec Domain of Interpretation (DOI) for Internet Security Association and Key Management Protocol (ISAKMP)”, December 2005 (Standards Track)
- **RFC 4306, “Internet Key Exchange (IKEv2) Protocol”, December 2005 (Standards Track)**
- **RFC 4307, “Cryptographic Algorithms for Use in the Internet Key Exchange Version 2 (IKEv2)”, December 2005 (Standards Track)**
- **RFC 4308, “Cryptographic Suites for IPsec”, December 2005 (Standards Track)**
- RFC 4309, “Using Advanced Encryption Standard (AES) CCM Mode with IPsec Encapsulating Security Payload (ESP)”, December 2005 (Standards Track)
- RFC 4312, “The Camellia Cipher Algorithm and Its Use With IPsec”, December 2005 (Standards Track)
- RFC 4430, “Kerberized Internet Negotiation of Keys (KINK)”, March 2006 (Standards Track)
- RFC 4322, “Opportunistic Encryption using the Internet Key Exchange (IKE)”, December 2005 (Informational)
- RFC 4434, “The AES-XCBC-PRF-128 Algorithm for the Internet Exchange Protocol (IKE)”, February 2006 (Standards Track)
- **RFC 4494, “The AES-CMAC-96 Algorithm and Its Use with IPsec”, June 2006 (Standards Track)**
- RFC 4543, “The Use of Galois Message Authentication Code (GMAC) in IPsec ERP and AH”, May 2006 (Standards Track)
- **RFC 4555, “IKEv2 Mobility and Multihoming Protocol (MOBIKE)”, June 2006 (Standards Track)**

- RFC 4615, “The Advanced Encryption Standard-Cipher-based Message Authentication Code-Pseudo-Random Function-128 (AES-CMAC-PRF-128) Algorithm for the Internet Key Exchange Protocol (IKE)”, August 2006 (Standards Track)
- **RFC 4718, “IKEv2 Clarifications and Implementations Guidelines”, October 2006 (Informational)**
- RFC 4807, “IPsec Security Policy Database Configuration MIB”, March 2007 (Standards Track)
- **RFC 4809, “Requirements for an IPsec Certificate Management Profile”, February 2007 (Informational)**
- **RFC 4835, “Cryptographic Algorithm Implementation Requirements for Encapsulating Security Payload (ESP) and Authentication Header (AH)”, April 2007 (Standards Track)**
- **RFC 4868, “Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 with IPsec”, May 2007 (Standards Track)**
- **RFC 4869, “Suite B Cryptographic Suites for IPsec”, May 2007 (Informational)**
- RFC 4877, “Mobile IPv6 Operation with IKEv2 and the Revised IPsec Architecture”, April 2007 (Standards Track)
- RFC 4891, “Using IPsec to Secure IPv6-in-IPv4 Tunnels”, May 2007 (Informational)
- RFC 4894, “Use of Hash Algorithms in Internet Key Exchange (IKE) and IPsec”, May 2007 (Informational)
- **RFC 4945, “The Internet IP Security PKI Profile of IKEv1/ISAKMP, IKEv2, and PKIX”, August 2007 (Standards track)**
- RFC 5265, “Mobile IPv4 Traversal across IPsec-Based VPN Gateways”, June 2008 (Standards Track)
- RFC 5374, “Multicast Extensions to the Security Architecture for the Internet Protocol”, November 2008 (Standards Track)
- RFC 5386, “Better-Than-Nothing Security: An Unauthenticated Mode of IPsec”, November 2008 (Standards Track)
- RFC 5406, “Guidelines for Specifying the Use of IPsec Version 2”, February 2009 (Best Current Practice)
- RFC 5529, “Modes of Operation for Camellia for Use with IPsec”, April 2009 (Standards Track)
- RFC 5566, “BGP IPsec Tunnel Encapsulation Attribute”, June 2009 (Standards Track)
- RFC 5660, “IPsec Channels: Connection Latching”, October 2009 (Standards Track)
- RFC 5755, “An Internet Attribute Certificate Profile for Authorization”, January 2010 (Standards Track)

6.1.2 – Security Association, Security Association Database and Security Parameter Index

A *Security Association* (SA) is a collection of which protocols and algorithms to use for authentication and encryption, together with the keys used, for communication in one direction between two IPsec enabled nodes (e.g. from Alice to Bob). A different SA is created for communication in the other direction, between the same two nodes (e.g. from Bob to Alice). Each pair of communicating IPsec nodes require an SA for each direction that data will be sent between them (normally both directions). Each node stores both SAs for a connection in its Security Association Database (SADB). It refers to the SA for outgoing traffic when it sends packets, and the SA for incoming traffic when it receives packets. When a secure connection is set up the first time (or if anything is changed) then the relevant Security Associations are negotiated and stored.

A *Security Association Database* (SADB) is a collection of Security Associations, on a given node. Each IPsec enabled node has its own SADB (this is not stored in a central DBMS, and in fact does not resemble what most people would call a Database – it's really more of a simple table). As the node negotiates Security Associations with other nodes, it stores each new one into its SADB.

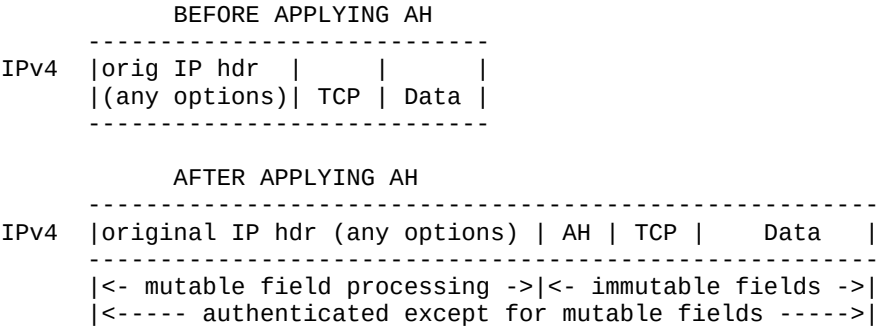
The *Security Parameter Index* (SPI) is an index into the SADB. An SPI together with a destination IP address uniquely identifies a particular security association.

6.1.3 – IPsec Transport Mode and IPsec Tunnel Mode

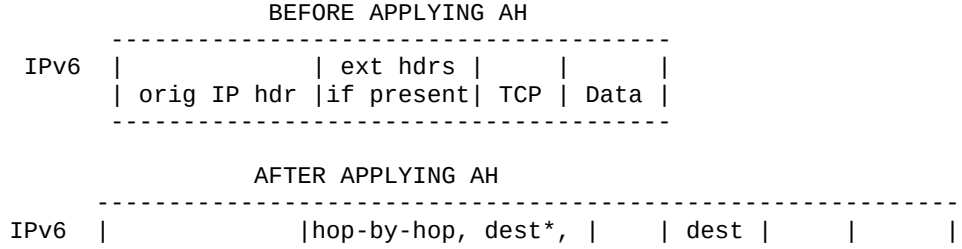
There are two *modes* in which a given IPsec connection can operate: *Transport Mode* and *Tunnel Mode*.

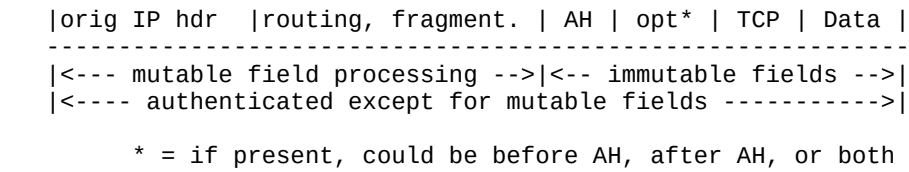
In *Transport Mode*, with AH, only the packet payload plus certain fields in the packet headers (including source and destination IP addresses and source and destination port numbers) are authenticated. This means the contents of the payload and those header fields are included in the calculation of the AH cryptographic checksum using HMAC. None of those fields are modified in any way by IPsec. Therefore the original addresses are used for routing of the packet. However, if NAT modifies any of these header fields as the packet goes through a NAT gateway (which is the normal behavior of NAT) then the cryptographic checksum will fail when the packet is received (after all, someone has tampered with the packet contents).

In IPv4 Transport Mode, the AH packet header is inserted after the IP header, but before the TCP (or UDP) header, as follows:



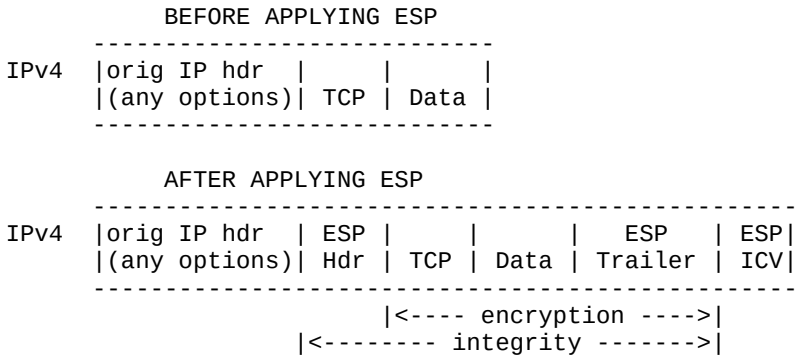
In IPv6 Transport Mode, the original IP packet header comes first, followed by one or more extension headers, one of which is the new AH header, followed by the TCP (or UDP) header, then data field (payload).



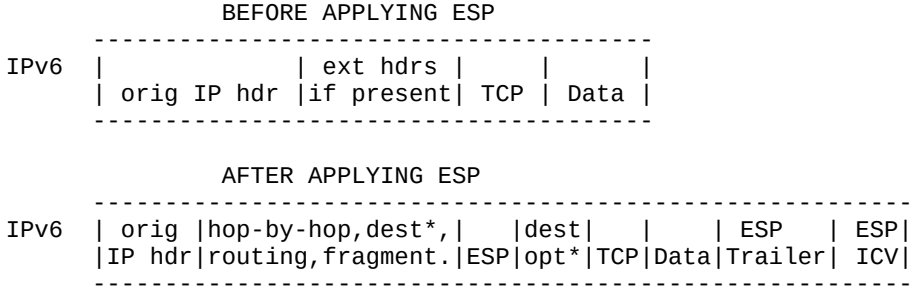


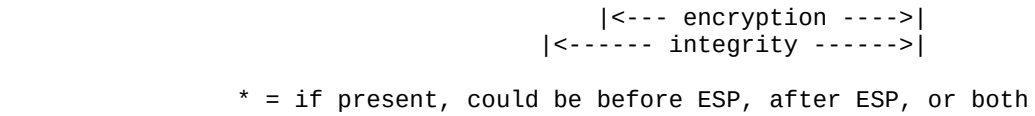
In *Transport Mode* with ESP, only the TCP (or UDP) header and the data field (payload) are encrypted. No other header fields are encrypted, or the packet could not be delivered. Transport mode is used only for host-to-host communications. No IPsec gateway is required. Any node involved in a Transport Mode IPsec connection must have support for IPsec Transport Mode. If automated key exchange is to be used, those nodes must also support a common Key Exchange protocol (IKEv1, IKEv2 or KINK). If available, IKEv2 is preferred. If IKE is used, each node requires an appropriate IPsec digital certificate that binds the public key to their IP address(es). If KINK is used, then a Kerberos Key Distribution Center must be available to all nodes using it.

In IPv4 Transport Mode with ESP, the original IP header comes first, followed by the ESP header, followed by the TCP (or UDP) header, followed by the data. With ESP, after the data field, there is an *ESP Trailer* and an *Integrity Check Value* (ICV). Encryption is done on the TCP (or UDP) header, the Data field (payload) and the ESP Trailer. Integrity (for the ICV) covers those fields plus the ESP header.



In IPv6 Transport Mode with ESP, the original IP header comes first, followed by one or more extension headers, one of which is the ESP extension header, followed by the original TCP (or UDP) header, then the data field (the packet payload). As with IPv4, there is an *ESP trailer*, and an *ESP ICV*. Encryption is done on any extension headers after the ESP extension header, the TCP (or UDP) header, the Data field (payload) and the ESP Trailer. Integrity (for the ICV) covers those fields plus the ESP header.

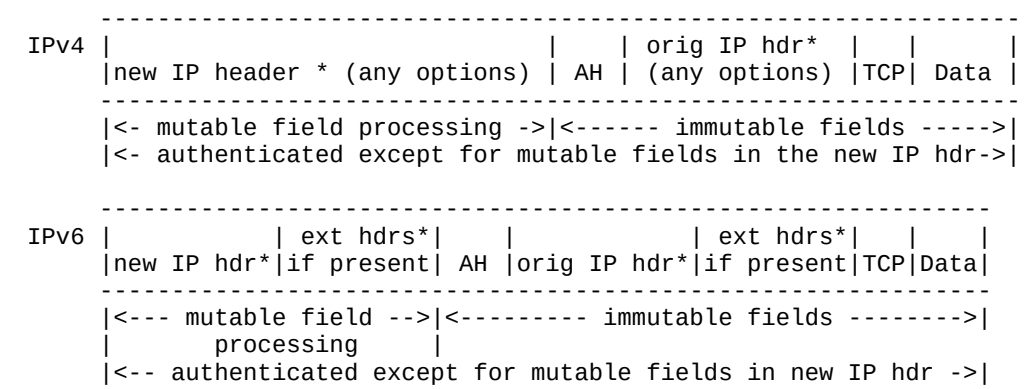




In *Tunnel Mode* (with AH and/or ESP), the entire original IP packet (all headers plus the payload) are encrypted and/or authenticated. The result is encapsulated into a new IP packet with new headers. This encapsulation is added to packets on the way out by an IPsec tunnel gateway, and removed on the way at the other end of the network path by another IPsec tunnel gateway. In between, it looks like a normal IPv4 (or IPv6) packet that has an odd looking payload, and is routed like any other packet. The IP version of the inner packet does not have to be the same as the IP version of the outer packet. You can tunnel IPv6 packets over IPv4, or IPv4 packets over IPv6. The node you connect to must support the version(s) of IP you send to it, though. After the encapsulation is removed and the authentication and/or encryption are removed, the resulting packet is forwarded to the inside of the tunnel gateway, where it continues on to its destination.

In IPv4 Tunnel Mode with AH, the outer IP header comes first (possibly with options), followed by the new AH header, and followed by the original entire packet.

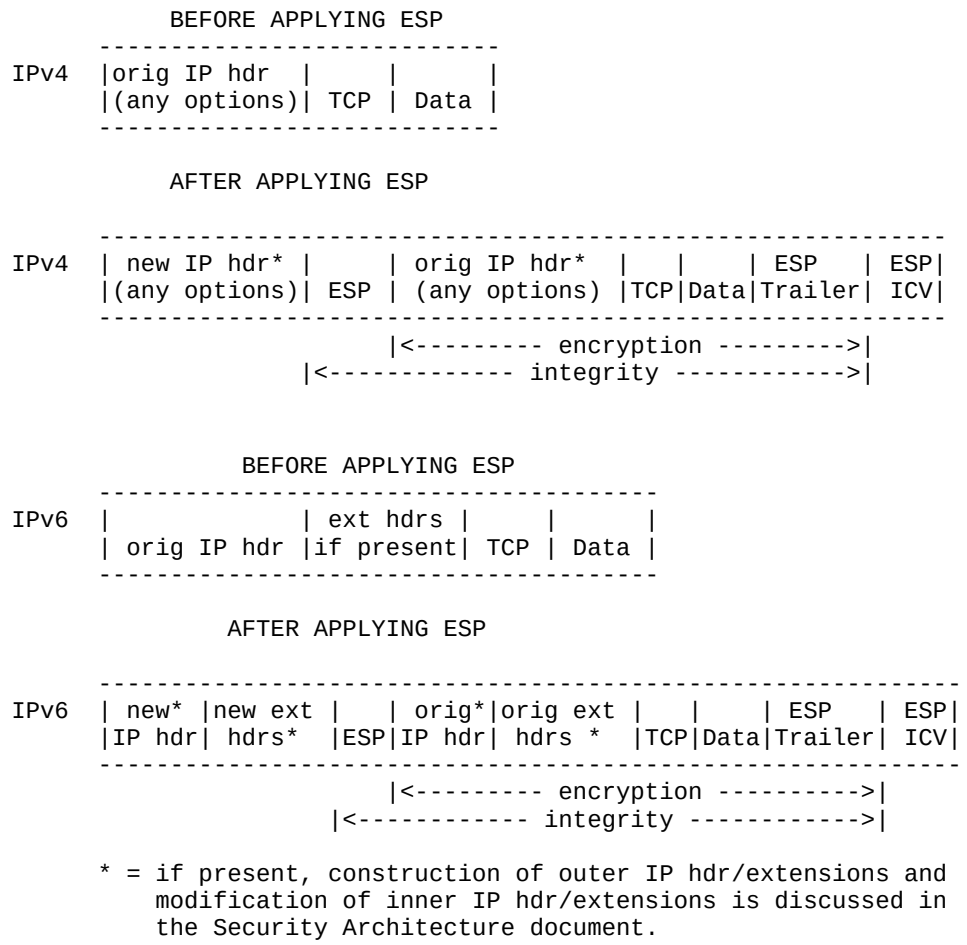
In IPv6 Tunnel Mode with AH, the outer IP header comes first, followed by one or more extension headers, including the AH extension header, followed by the original entire packet (which itself may contain extension packet headers, but they won't be processed until after the packet is de-tunneled).



In IPv4 Tunnel Mode with ESP, the outer IP header comes first, followed by the ESP header, followed by any options from the original packet header, followed by the TCP (or UDP) header, followed by the data field (payload). Again, the data is followed by an *ESP Trailer* and *ESP ICV*. Encryption covers everything after the ESP header, up to and including the ESP trailer. The integrity value covers all of that plus the ESP header.

In IPv6 Tunnel Mode with ESP, the outer packet IP packet header comes first followed by any new packet header extensions, followed by the ESP header, then the entire original packet. As before, the data field (payload) is now followed by the *ESP trailer* and the *ESP ICV*. Encryption covers everything

after the ESP header, up to and including the ESP trailer. The integrity value covers all of that plus the ESP header.



The IPsec tunnel processing can be physically located inside a gateway firewall or router, or it can be a in a node that does *just* IPsec tunneling, on the inside of an existing router or firewall. All IPsec related processing (generating or validating the HMAC cryptographic checksum and/or packet encryption/decryption) takes place in the tunneling IPsec node. IPsec Tunnel Mode is primarily used for network-to-network tunnels, but it *can* be used for host-to-network communications (e.g. a road warrior connecting into the home network securely), or even host-to-host communications (e.g. for private chat or VoIP). If any hosts (as opposed to gateways) are involved in an IPsec Tunnel Mode connection, those hosts would need to support IPsec tunnel mode. If the tunnel is built between two gateway nodes (network-to-network tunnel), then any other node in either network can send things through that tunnel to nodes in the other network without having to know anything about IPsec. If automated key exchange is to be used, the participating nodes must also support a common Key Exchange protocol (IKEv1, IKEv2 or KINK). If available, IKEv2 is preferred. If IKE is used, each node requires an appropriate IPsec digital certificate that binds the public key to their IP address(es), for mutual authentication. If KINK is used, then a Kerberos Key Distribution Center must be available to all nodes using it.

An IPsec tunnel must be set up in the sending and receiving nodes so that the sending node knows what address to put in the *outer packet header*. This configuration would specify things like "all traffic destined for 123.45.56.00/24 is to be tunneled and sent to the gateway located at 123.45.67.1", or "Accept IPsec tunneled traffic from the node which is located at 87.65.34.21, de-tunnel it, and route the inside packet onto the LAN". Regardless of what transport the original traffic used (TCP or UDP), the tunneled traffic will be over UDP. This introduces additional overhead, and can complicate the built-in error recovery mechanism in TCP. The IP addresses in the outer packet header are not authenticated.

If IPv4 tunneled traffic crosses any router, the addresses of all nodes musts be valid global unicast IP addresses (although this could be simulated using BINAT with a node that has a private IP address behind a NAT gateway). It would be possible to create an IPsec path over IPv4 entirely within a routing domain using private IP addresses. Basically there must be a flat address space (or a reasonable facsimile thereof) over the entire path of the connection. A classic problem with VPNs connecting nodes in disjoint private networks is that the private address spaces in the two networks must not *overlap*. If two companies are both using 10.0.0.0/8 as their private addresses, either VPNs will not work between them, or at least one network must renumber (e.g. to 172.16.0.0/12). If you have an HQ network and several branch offices, you might want to use 10.0.0.0/16 for HQ, 10.1.0.0/16 for the first branch office, 10.2.0.0/16 for the second branch office, etc. This would allow up to 256 branch offices, each with up to 65,535 nodes. This way, if you do want to build VPNs between them someday, there will not be any overlap.

6.1.4 – IPsec over IPv6

In IPv6, none of the issues related to NAT arise, since all nodes can easily obtain global unicast IPv6 addresses, and the entire world is a single flat address space. IPsec works *beautifully* over IPv6, and this is one of the strongest arguments for migrating to Dual Stack sooner rather than later. Use the IPv6 side for protocols that require a flat address space, such as IPsec, SIP, P2P, etc. Use the IPv4 side for legacy applications like web surfing, E-mail, etc. You can gradually move those over to IPv6 as well.

The IPv6 packet header design supports IPsec very well. There are two packet header extensions defined, one for AH and one for ESP. The AH extension header will be inserted if and only if AH is used on that packet. For details, see RFC 4302, "IP Authentication Header". The ESP extension header will be inserted if and only if ESP is used on that packet. For details, see RFC 4303, "IP Encapsulating Security Payload (ESP)".

6.1.5 – IPsec in Multicast Networks

It is possible to deploy IPsec in a multicast network. Security Association negotiation is rather more complicated in a one-to-many connection than in a one-to-one connection. RFC 5374, "Multicast Extensions to the Security Architecture for the Internet Protocol" covers the details. IPsec in multicast networks could allow content providers to control access to multicast content using ESP and some clever key management. For example, each valid subscriber could be issued a unique IPsec digital certificate (tied to their set top box) that would allow decrypting the symmetric session key used to encrypt the content. If they don't pay their bill, their certificate could be revoked.

6.1.6 – Using IPsec to secure L2TP Connections

L2TP (Layer 2 Tunneling Protocol) itself does not provide either privacy or authentication. RFC 3193, "Securing L2TP using IPsec" specifies how to use IPsec in conjunction with it to add privacy (with ESP) and/or authentication (with AH) to an L2TP based system.

6.2 – Internet Key Exchange (IKE)

The Internet Key Exchange (IKEv1 and IKEv2) is based on ISAKMP (Internet Security Association and Key Management Protocol), which is a *framework* for key exchange. It uses parts of the Oakley and SKEME (Secure Key Exchange MEchanism for Internet) protocols within this framework. Oakley describes a series of key exchanges, known as modes, and specifies things such as *perfect forward secrecy* for keys, identity protection and authentication. It is discussed in RFC 2412, "The Oakley Key Determination Protocol". SKEME is a key exchange technique which provides anonymity and quick key refreshment. There is no RFC that covers SKEME, but there are some papers available online. There is coverage of the parts of Oakley and SKEME used in IKE in RFCs 2408 and 2409.

IKE uses the Diffie-Hellman Key Agreement protocol (see Appendix A) to securely exchange a shared secret, from which symmetric session keys for AH and ESP are derived. IKE is also used to mutually authenticate nodes to each other. Authentication can be accomplished with a pre-shared secret (manually distributed to each node), or by use of IPsec digital certificates (ones that bind IPv4 and/or IPv6 addresses to the public key). Each pair of nodes that use IPsec use IKE to do key exchange and mutual authentication, which results in setting up a Security Association (SA) at each end for the other node in the pair.

IKE is usually implemented as a daemon process (a software application that starts running when the computer boots, and stays running until shutdown) on each IPsec enabled node, in *user space* (the part of memory where user applications run). IKE communicates via UDP over port 500. There is no client or server role, the communication is between peers. Either node of a pair can initiate an IKE connection. The other node of the pair will accept it. The AH and ESP packet processing is embedded in the TCP/IP stack (specifically in the IP layer), which usually runs in *kernel space* (the part of memory where the operating system kernel runs, typically protected from access by user applications).

Note in the following that Oakley defines *modes*: Main Mode, Aggressive Mode and Quick Mode. ISAKMP defines *phases*: Phase 1 and Phase 2. Main Mode and Aggressive Mode take place during ISAKMP's Phase 1, while Quick Mode takes place during ISAKMP's Phase 2.

IKE Phase 1 establishes an encrypted, authenticated communication channel between the two parties. It first uses the Diffie-Hellman Key Agreement protocol, which produces a shared secret. A symmetric session key is derived from the shared secret by both parties, which is used to encrypt further IKE exchanges. The goal of Phase 1 is to create a *bidirectional* ISAKMP Security Association (SA). Mutual Authentication can be accomplished using either a pre-shared secret, or via cryptographic challenge/response using IPsec public key digital certificates. Either a pre-shared secret, or an IPsec certificate must be installed on each IPsec enabled node when the system is deployed. Phase 1 can

operate in either *Main Mode* or *Aggressive Mode*. Main Mode protects the identities of the nodes, but takes longer. Aggressive mode is faster, but does not protect the identities of the nodes.

IKE Phase 2 uses the secure channel established in Phase 1 to negotiate additional Security Associations (SAs) for services such as IPsec. The output of Phase 2 is a pair of *unidirectional* SAs (one for traffic from Alice to Bob and one for traffic from Bob to Alice). On each node, one of these SAs is used for inbound traffic and the other one for outbound traffic. Phase two operates in *Quick Mode*.

Cryptographic challenge/response is used by each end to authenticate itself to the other. This is based on public/private key pairs (asymmetric cryptography), and digital certificates, which are described in Appendix A.4. Each direction works as follows (flipping roles A and B when the second direction is done):

> *Step 1* –Node A sends its public key digital certificate to Node B.
>
> *Step 2* –Node B verifies Node A's digital certificate by checking its digital signature, its expiration date, and its revocation status. It also climbs the chain of trust to a trusted root key. If the identity in the certificate is an IP address, it must match the source IP address of the IKE connection.
>
> *Step 3* –Node B generates a random string and encrypts it using Node A's public key (from its digital certificate) and sends it as a *challenge* to Node A.
>
> *Step 4* –Node A decrypts the *challenge* using its own private key and returns the result to Node B as its *response*.
>
> *Step 5* –Node B compares the *response* to the original string. If they match, that is proof that Node A possesses the private key associated with the public key in Node A's digital certificate (without Node A revealing its private key to anyone). This authenticates Node A to Node B.

An IPsec digital certificate is like a Server (SSL) digital certificate or a Client digital certificate. The primary difference is what identity information the public key is bound to. In a *Server* cert, the identity information is an FQDN (e.g. www.example.com) and an organization name (in the distinguished name). In a *Client* cert, the identity information is a person's name and E-mail address (in the distinguished name). It is also possible for an IPsec certificate to specifically include IKE as a valid key usage (id_kp_ipsecIKE attribute). In an IPsec cert, there are several possibilities for the identity:

- Individual IPv4 and/or IPv6 address (ID_IPV4_ADDR and ID_IPV6_ADDR). These do not work well if the connection traverses NAT. There is no problem with using IPv6 addresses.
- IPv4 or IPv6 subnet (ID_IPv4_ADDR_SUBNET and ID_IPv6_ADDR_SUBNET). The same issues are involved if the connection traverses NAT. For example, you could identify your node as being in the subnet *2001:418:5403:3000::/64*.
- IPv4 or IPv6 address range (ID_IPv4_ADDR_RANGE and ID_IPV6_ADDR_RANGE). This is used for specifying a block of addresses that doesn't happen to fall on (power of 2) subnet boundaries (e.g. all addresses from *2001:418:5403:3000::100* to *2001:418:5403:3000::200*).
- FQDN (ID_FQDN). This depends on trusting the mapping from FQDN to IP address; hence DNS should only be used if DNSSEC is deployed. Otherwise the resolution from FQDN to IP address

must be handled by some other means, which *is* trusted. Again, NAT can cause problems with this.

If IP addresses are specified, during the authentication process the source IP address of the IKE connection must match the IP address (byte for byte) in the IPsec digital certificate. This is similar to SSL/TLS, where the node name you are connecting to must match the FQDN in the server certificate. In SSL, if you connect to an alias name or a numeric IP address, you will get an error.

6.2.1 – Internet Key Exchange version 2 (IKEv2)

IKEv1 was defined in November 1998. There are still some IPsec implementations that support only IKEv1, so some IKEv2 implementations also support IKEv1 for backward compatibility (e.g. RACOON2 in BSD). There were many issues with IKEv1, which led to the creation of IKEv2 in December 2005. The issues with IKEv1 include the following:

- The specification of IKEv1 was spread over three basic RFCs (2407, 2408 and 2409), plus others for NAT traversal (3715), etc. In comparison, almost all of IKEv2 was specified in RFC 4306. The list of supported cryptographic algorithms was split off into RFC 4307 for ease of updating the algorithm suite in the future.
- IKEv1 had no support for SCTP or Mobile IP protocols. Both are supported in IKEv2. For SCTP, see RFC 3554, "On the Use of Stream Control Transmission Protocol (SCTP) with IPsec". For Mobile IP, See RFC 4877, "Mobile IPv6 Operation with IKEv2 and the Revised IPsec Architecture" (Section 7.3), and RFC 4555, "IKEv2 Mobility and Multihoming Protocol (MOBIKE)".
- IKEv1 had *eight* distinct initial exchange mechanisms, each of which had advantages and disadvantages, vs. *one* four-message initial exchange in IKEv2.
- IKEv1 included an excessive number of cryptographic algorithms which resulted in complex implementation and long (and costly) certification processes (e.g. Common Criteria and FIPS 140-2). For details, see RFC 2409, updated by RFC 4109, "Algorithms for Internet Key Exchange version 1 (IKEv1)". In comparison, IKEv2 reduced the number of supported cryptographic algorithms. For details, see RFC 4306, "Cryptographic Algorithms for Use in the Internet Key Exchange Version 2 (IKEv2)".
- IKEv1 had reliability issues due to poor state management. This could result in a hung node, requiring Dead Peer Detection (which was never standardized, leading to interoperability issues). IKEv2 added sequence numbers and acknowledgements to greatly improve state management.
- IKEv1 had issues with Denial of Service attacks, where connections from spoofed addresses could cause it to do *expensive* asymmetric key processing. IKEv2 does such processing only *after* it verifies the validity and existence of a client.
- The IPv6 Ready test centers have no certification tests for IKEv1, but they do for IKEv2.They also have certification tests for IPsec over IPv6.

Some IPv4 routers (especially ones for home or small office) include an *IPsec Helper* function. This routes all IPsec traffic to the first node that negotiates a Security Association (SA). The assumption is that there is only one IPsec endpoint inside the home network. This is yet another attempt to make IPsec work over NAT. In IPv6 routers, there is no need for any IPsec Helper function.

Another issue with NAT involves fragmentation. During IKE, if an IPsec digital certificate is sent, this often is larger than a single packet, which leads to packet fragmentation. Many NAT gateways will simply drop fragmented packets, as these are usually part of a hacking attack. This is never a problem in IPv6.

It should be obvious by now why there are so many problems using IPsec in the legacy First Internet. NAT helped keep the Internet going while IPv6 was being created, but now that IPv6 is complete and available, we should switch to it *at least* for certain protocols (especially IPsec and SIP) rather than create ever more complex workarounds to fix the problems caused by NAT.

6.2.3 – Kerberized Internet Negotiation of Keys - KINK

There is an alternative to IKE for securely exchanging keys among IPsec enabled nodes, called *KINK*. It is defined in RFC 4430, "Kerberized Internet Negotiation of Keys (KINK)". With IKE, each node needs an IPsec digital certificate to authenticate itself to other nodes, which is done on a peer-to-peer basis. A full PKI must be deployed to support the issuance and maintenance of these certificates. With KINK, nodes need only mutually authenticate with the Authentication Server of the Key Distribution Center (KDC) of a Kerberos facility. No IPsec digital certificates are required for each node, and no PKI is required. However, there must be a Kerberos KDC that can do the necessary authentication, and all participating IPsec enabled nodes need client-side support for Kerberos and KINK. Deploying Kerberos securely can be just as big a challenge as deploying a PKI. If your nodes will need to connect over IPsec to nodes in other organizations, KINK is probably not the best way to go. IKE is clearly the preferred Key Exchange technology.

6.3 – Mobile IP

With IPsec, the protocols for IPv4 and IPv6 are fairly similar, and for the most part, there is a single set of standards that cover both IPv4 and IPv6. The main difference is the almost universal presence of NAT in IPv4 and total absence of it in IPv6. With Mobile IP, on the other hand, the support for IPv4 and IPv6 are quite different, each has its own set of standards, and they continue to diverge more and more over time. Mobile IPv6 is much more capable than Mobile IPv4, and due to the imminent exhaustion of the IPv4 address space, most new research is being done on Mobile IPv6, not Mobile IPv4. Again, the biggest problem with Mobile IPv4 is the presence of NAT in most real-world IPv4 networks.

In Mobile IP, the *mobile node* is the one that is moving around, and will try to access the Internet via different networks at different times. The *correspondent node* is the node that the mobile node is trying to connect to (perhaps a server). A *home agent* is a special router (that supports Mobile IP) in the home network of the mobile node, which accepts tunneled connections from the mobile node and relays them onward as if the mobile node was *in* the home network.

Mobile IPv4 requires that all nodes (mobile node, correspondent node and home agent) support IPv4 (it doesn't matter whether or not any of them also support IPv6). Mobile IPv4 is basically an application of tunneling, and the tunnels are "4in4" (IPv4 tunneled over IPv4). There is no support at all for IPv6 in Mobile IPv4. If you plug your node into an IPv6-only foreign network, Mobile IPv4 will not work.

Mobile IPv6 requires that all nodes (mobile node, correspondent node and home agent) support IPv6 (it doesn't matter whether or not any of them also support IPv4). Mobile IPv6 is basically an application of tunneling, and the tunnels are "6in6" (IPv6 tunneled over IPv6). There is no support at all for IPv4 in Mobile IPv6. If you plug your node into an IPv4-only foreign network, Mobile IPv6 will not work. Mobile IPv6 has one important aspect that is completely missing in Mobile IPv4, which is *route optimization*. It is likely that the mobile node may have a more direct (shorter) path to the correspondent node than via the home agent, and Mobile IPv6 provides a mechanism to help find and switch over to that better path. Mobile IPv4 has no such mechanism.

There are efforts underway to produce a Mobile IP that can work with IPv4 and/or IPv6, on dual stack nodes, and use 4in6 or 6in4 tunnels, as needed. It may include 6to4 and 4to6 translation as well. So far this is still in the early days. There is a start on this in RFC 5555. There are *many* difficult issues involved in making this work in general.

Until "dual stack" Mobile IP evolves, the worlds of Mobile IPv4 and Mobile IPv6 are pretty much disjoint. If you have a dual stack node, and want to go mobile, you would need two complete (and independent) Mobile IP systems (client, home agent, tunnel scheme, etc.) – one for IPv4 and one for IPv6. Even if you deployed such an infrastructure, only the Mobile IPv6 system would be able to optimize routes.

Both Mobile IPv4 and Mobile IPv6 require extensions in the TCP/IP stack to work, which are not normally included in a base distribution today. For example, Windows 7 includes the extensions to work as a Correspondent Node, but not as a Mobile Node. Few routers include the extensions to work as a Home Agent. In general, there is very little in the way of commercial or open source software that includes support for Mobile IPv4 or Mobile IPv6 at the present. This is really "beyond the bleeding edge" at this point in time. Due to incompatibilities with NAT (and the need for all addresses to be globally routable), it is not likely that many vendors will be willing to do much product development on Mobile

IPv4. The absence of NAT in IPv6 (together with the presence of route optimization in Mobile IPv6) may increase the probability of products being developed based on Mobile IPv6.

The best information on Mobile IPv6 I've found is in the book "Mobile IPv6: Protocols and Implementation" by Qing Li, Tatuya Junmei and Keiichi Shima (three of the developers from the Kame project). This is published by Morgan Kaufmann.

6.3.1 – Mobile IPv4

The following RFCs are related to Mobile IPv4:

- RFC 2290, "Mobile-IPv4 Configuration Option for PPP IPCP", February 1998 (Standards Track)
- RFC 2794, "Mobile IP network Access Identifier Extension for IPv4", March 2000 (Standards Track)
- RFC 2977, "Mobile IP Authentication, Authorization, and Accounting Requirements", October 2000 (Informational)
- **RFC 3024, "Reverse Tunneling for Mobile IP, revised", January 2001 (Standards Track)**
- **RFC 3344, "IP Mobility Support for IPv4", August 2002 (Standards Track)**
- **RFC 3519, "Mobile IP Traversal of Network Address Translation (NAT) Devices", April 2003**
- RFC 3543, "Registration Revocation in Mobile IPv4", August 2003 (Standards Track)
- RFC 3846, "Mobile IPv4 Extensions for Carrying Network Access Identifiers", June 2004 (Standards Track)
- RFC 3957, "Authentication, Authorization and Accounting (AAA) Registration Keys for Mobile IPv4", March 2005 (Standards Track)
- RFC 4004, "Diameter Mobile IPv4 Application", August 2005 (Standards Track)
- RFC 4058, "Protocol for Carrying Authentication for Network Access Requirements (PANA)", May 2005 (Informational)
- RFC 4064, "Experimental Message, Extensions and Error Codes for Mobile IPv4", May 2006 (Standards Track)
- **RFC 4093, "Problem Statement: Mobile IPv4 Traversal of Virtual Private Network (VPN) Gateways", August 2005 (Informational)**
- RFC 4433, "Mobile IPv4 Dynamic Home Agent (HA) Assignment", March 2006 (Standards Track)
- RFC 4636, "Foreign Agent Error Extension for Mobile IPv4", October 2006 (Standards Track)
- RFC 4721, "Mobile IPv4 Challenge/Response Extensions (Revised)", January 2007 (Standards Track)
- RFC 4977, "Problem Statement: Dual Stack Mobility", August 2007 (Informational)
- RFC 4917, "Mobile IPv4 message String Extension", June 2007 (Standards Track)
- RFC 5030, "Mobile IPv4 RADIUS Requirements", October 2007 (Informational)
- **RFC 5177, "Network Mobility (NEMO) Extensions for Mobile IPv4", April 2008 (Standards Track)**
- **RFC 5265, "Mobile IPv4 Traversal across IPsec-Based VPN Gateways", June 2008 (Standards Track)**
- RFC 5266, "Secure Connectivity and Mobility Using Mobile IPv4 and IKEv2 Mobility and Multihoming (MOBIKE)", June 2008 (Best Current Practices)
- RFC 5446, "Service Selection for Mobile IPv4", February 2009 (Informational)
- **RFC 5454, "Dual-Stack Mobile IPv4", March 2009 (Standards Track)**

6.3.2 – Mobile IPv6

The following RFCs are related to Mobile IPv6:

- **RFC 3775, "Mobility Support in IPv6", June 2004 (Standards Track)**
- RFC 3776, "Using IPsec to Protect Mobile IPv6 Signaling Between Mobile Nodes and Home Agents", June 2004 (Standards Track)
- **RFC 3963, "Network Mobility (NEMO) Basic Support Protocol", January 2005 (Standards Track)**
- RFC 4260, "Mobile IPv6 Fast Handovers for 802.11 Networks", November 2005 (Informational)
- RFC 4285, "Mobile Node Identifier Option for Mobile IPv6 (MIPv6)", November 2005 (Standards Track)
- **RFC 4285, "Authentication Protocol for Mobile IPv6", January 2006 (Informational)**
- RFC 4294, "IPv6 Node Requirements", April 2006 (Informational)
- RFC 4295, "Mobile IPv6 Management Information Base", April 2006 (Standards Track)
- RFC 4449, "Securing Mobile IPv6 Route Optimization Using a Static Shared key", June 2006 (Standards Track)
- RFC 4487, "Mobile IPv6 and Firewalls: Problem Statement", May 2006 (Informational)
- **RFC 4584, "Extension to Sockets API for Mobile IPv6", July 2006 (Informational)**
- RFC 4866, "Enhanced Route Optimization for Mobile IPv6", May 2007 (Standards Track)
- RFC 4877, "Mobile IPv6 Operation with IKEv2 and Revised IPsec Architecture", April 2007 (Standards Track)
- RFC 4882, "IP Address Location Privacy and Mobile IPv6: Problem Statement", May 2007 (Informational)
- RFC 4980, "Analysis of Multihoming in Network Mobility Support", October 2007 (Informational)
- RFC 5014, "IPv6 Socket API for Source Address Selection", September 2007 (Informational)
- RFC 5026, "Mobile IPv6 Bootstrapping in Split Scenario", October 2007 (Standards Track)
- RFC 5094, "Mobile IPv6 Vendor Specific Option", December 2007 (Standards Track)
- RFC 5096, "Mobile IPv6 Experimental Messages", December 2007 (Standards Track)
- RFC 5149, "Service Selection for Mobile IPv6", February 2008 (Informational)
- **RFC 5213, "Proxy Mobile IPv6", August 2008 (Standards Track)**
- RFC 5269, "Distributing a Symmetric Fast Mobile IPv6 (FMIPv6) Handover Key Using SEcure Neighbor Discovery (SEND)", June 2008 (Standards Track)
- RFC 5271, "Mobile IPv6 Fast Handovers for 3G CDMA Networks", June 2008 (Informational)
- RFC 5380, "Hierarchical Mobile IPv6 (HMIPv6) Mobility Management", October 2008 (Standards Track)
- RFC 5419, "Why the Authentication Data Suboption is Needed for Mobile IPv6 (MIPv6)", January 2009 (Informational)
- RFC 5447, "Diameter Mobile IPv6: Support for Network Access Server to Diameter Server Interaction", February 2009 (Standards Track)
- RFC 5380, "Hierarchical Mobile IPv6 (HIMPv6) Mobility Management", October 2008 (Standards Track)
- RFC 5488, "Network Mobility (NEMO) Management Information Base", April 2009 (Standards Track)

- **RFC 5555, "Mobile IPv6 Support for Dual Stack Hosts and Routers", June 2009 (Standards Track)**
- RFC 5568, "Mobile IPv6 Fast Handovers", July 2009 (Standards Track)
- RFC 5637, "Authentication, Authorization and Accounting (AAA) Goals for Mobile IPv6", September 2009 (Informational)
- RFC 5778, "Diameter Mobile IPv6: Support for Home Agent to Diameter Server Interaction", February 2010 (Standards Track)
- RFC 5779, "Diameter Proxy Mobile IPv6: Mobile Access Gateway and Local Mobility Anchor Interaction with Diameter Server", February 2010 (Standards Track)

6.3.3 – The Building Blocks of Mobile IP

As technology progresses, and the size and weight of our network capable devices continue to shrink, those devices are becoming more and more mobile. Desktop personal computers are more mobile than mainframes or minicomputers; notebook computers are more mobile than desktop computers; netbooks and tablet computers are more mobile than notebook computers; and smart phones and other consumer products with wireless connectivity (e.g. MP3 players) are extremely mobile. The connections can be via wired Ethernet or wireless (Wi-Fi, Wi-Max, etc). If the movement of your node only involves moving from one connection to another within a single network segment, there is no need for Mobile IP. Your node's Home Address should work regardless of which tap of your Home Network it is connected to within a single subnet. If your node is moving from one subnet to another, and you want to continue using the same IP address, then you need Mobile IP. If your node is continually moving (say in a car or train), and there is a series of wireless access points, each in a different subnet, you need Mobile IP, including fast handoff from one context to another. There are a number of components that are involved in a typical Mobile IP network scenario, whether we are talking Mobile IPv4 or Mobile IPv6. These include:

Mobile Node – a node that can move relative to its Home Network. The mobile node must have support for Mobile Node operation. Your Mobile Node must have an address valid on the Foreign Network to which it is connected (which is called the Foreign Address, and is used primarily as one end of an IP-in-IP tunnel to your Home Network), as well as an address on the home network (which is called the Home Address, and sends and receives packets only via the tunnel to the Home Network). No version of Windows currently has support for Mobile IP Mobile Node operation out of the box. Most Open Source operating systems do have this support at least to some level.

Home Address – the IP address of your Node that is valid (and unique) on your Home Network. This could be an IPv4 or IPv6 address depending on the version(s) of Mobile IP that your Node is using. Packets with this source address will not be routed onto the Foreign Network (where they would not work properly); they are pushed into the Mobile Node's end of the IP-in-IP tunnel, and come out at the Home Agent which routes them onto the Home Network. Likewise, packets from the Home Network addressed to the Mobile Node's Home Address will be accepted by the Home Agent and pushed into the Home Network end of the IP-in-IP tunnel. The packets will come out at the Mobile Node's end of the tunnel, where they will be sent to your Mobile Node. This Home Address is obtained directly from Mobile IP, and indirectly from DHCPv4, DHCPv6 or even Stateless Address Autoconfiguration on your Home Network. This address could also be manually configured.

Home Network – this is the network that your node (with its Home Address) would work in if it were not mobile. It is also the network in which your Home Agent lives.

Home Agent – there must be a router in your Home Network that has support for Mobile IP, and can and will accept a connection from an IP-in-IP tunnel from your Mobile Node. The Home Agent will relay packets from that tunnel onto the Home Network as if they had originated in your Home Network. The Home Agent also must accept packets addressed to your Home Address and push those into its end of the IP-in-IP tunnel, where they will be sent to your Mobile Node. It also must respond to ARP (IPv4) or ND (IPv6) queries to the Remote Node's Home Address (by proxy), e.g. using proxy ARP.

Foreign Network – the network to which the Mobile Node is now physically connected. Your Mobile Node must obtain an IP address (called the Foreign Address) that is valid (and not already in use) on the Foreign Network. This might be an IPv4 or IPv6 address, depending on what IP version the foreign network is running. Packets from your Mobile Node, with a source address which is your Home Address are pushed into the IP-in-IP tunnel by Mobile IP and come out at your Home Agent, where they are routed onto the Home Network as if your Node were physically present there. Packets from your Home Network that are addressed to your Home Address are accepted by the Home Agent which pushes them into its end of the IP-in-IP tunnel, and they come out into your Mobile Node, which accepts them.

Foreign Address – the IP address that is valid on the foreign network, which is obtained by your Mobile Node, and which it uses only as the foreign end of an IP-in-IP tunnel, the other end of which is at your Home Agent. This could be an IPv4 or IPv6 address depending on the version(s) of IP supported on the Foreign Network. The Mobile IP code on your node will obtain this address via DHCPv4, DHCPv6 or Stateless Address Autoconfiguration. The Foreign address is used primarily as one end of an IP-in-IP tunnel.

Correspondent Node – this is the node your Mobile Node is connecting to. It could be a fixed node or possibly another Mobile Node. The TCP/IP stack on that node must include support for *Mobile IP Correspondent Node* operation. This node must be reachable from your Home Network. Windows Vista and Windows 7 have support for *Mobile IP Correspondent Node* operation. With Mobile IPv6, once the basic routed-via-tunnel communication path is working, route optimization will attempt to find a way to transmit packets directly from the Mobile Node to the Correspondent Node (as opposed to going through the Home Agent).

Foreign Agent – there is an optimization for situations where there might be a number of Mobile Nodes in a single Foreign Network. It involves deploying a Foreign Agent. This functionality is implemented in a router in the Foreign Network. It actually builds the tunnel to the Home Agent, and the Mobile Nodes get a free ride over that tunnel. The Mobile Nodes send packets to the Foreign Agent, which tunnels them to the Home Agent, which then detunnels the packets and sends them as if they had originated in the Home Network. Packets directed to any Mobile Node are accepted by the Home Agent (via proxy ARP), then tunneled to the Foreign Agent. The Foreign Agent then sends the packets (native) to the Mobile Node. The advantage here is only one tunnel is required from the Foreign Network to the Home Network.

6.3.4 – Implementations

There is an Open Source project called “Dynamics Mobile IP” for Linux. It appears to have reached a pre-release version (0.81) in October 2001, and gone quiet. It appears to support only Mobile IPv4.

Another Open Source project called “Transparent Mobile IP” reached version 0.5a in 2003, then went quiet. It also appears to be Mobile IPv4 only.

Sun implemented parts of Mobile IPv4, including Home Agent and Foreign Agent functionality (no Mobile Node or Correspondent Node functionality), for the Solaris 10 operating system. It was removed after Solaris 10 8/07.

An implementation of Mobile IPv6 for Linux (kernel 2.4.22) was done at Helsinki University of Technology’s MIPL project. It includes support for Mobile Node and Home Agent functionality, and involves applying a patch to the Linux Kernel.

Microsoft implemented the Correspondent Node functionality in Windows Vista and Windows 7, but not the Mobile Node functionality.

The most recent Mobile IPv6 work on FreeBSD appears to have been done by the Kame project before they shut down, and the download is dated 2001. It was based on the FreeBSD 4.2 operating system. There is a discussion of how to deploy and configure Mobile IPv6 in the “Mobile IPv6” book previously mentioned, using the Kame IPv6 stack. The instructions do not appear to be aimed at novices.

6.3.4 – Conclusions on Mobile IP

Since Mobile IP is not widely deployed, and incompletely implemented, that’s all we are going to cover about it for now. In future editions of this book, hopefully it will be necessary to cover Mobile IPv6 in greater detail, because it will have matured significantly and be widely deployed. For now see the book on Mobile IPv6 mentioned above.

Chapter 7 – Transition Mechanisms

This chapter covers a variety of protocols and mechanisms that were created to simplify the introduction of IPv6 into the Internet. The goal is not to make an abrupt transition from all-IPv4 to all-IPv6 on some kind of "flag day". That would be unbelievably disruptive, and unlikely to succeed. The goal is to gradually add new capabilities that take advantage of IPv6, or work far better over it (e.g. IPsec VPN, SIP, Mobile IP, IPTV and most other multicast), while continuing to use IPv4 for those things that work tolerably well via over IPv4 with NAT (e.g. web, email, ftp, ssh, and most client-server). This allows immediate alleviation of the most grievous problems caused by widespread deployment of NAT and other shortcomings of IPv4, while allowing a longer, more controlled migration of those protocols that do not benefit as much from IPv6. Eventually, all protocols and applications will be migrated, and IPv4 can quietly be dropped from operating systems and hardware. However, this will probably be 5 to 10 years from now. As more and more applications are transitioned to IPv6, that will take the pressure off of the remaining stock of IPv4 addresses.

Most of these transition mechanisms are defined in RFCs, and were designed as a part of the IPv6 standard. There are many mechanisms, some with confusingly similar names, such as "6in4", "6to4", and "6over4", which are all quite different. Most deployments of IPv6 will use one or more of these transition mechanisms; none will use all of them. Some of the transition mechanisms are designed for use in the early phases of the transition, where there is an "ocean" of IPv4 with small (but growing) islands of IPv6 (e.g. 6in4 tunneling). Some are for use in the later stages of the transition, where the Internet has flipped into an "ocean" of IPv6, with small (and shrinking) islands of IPv4 (e.g. 4in6 tunneling).

7.1 – Relevant Standards

The following standards are relevant to IPv6 transition mechanisms:

- RFC 2473, "Generic Packet Tunneling in IPv6 Specification", December 1998 (Standards Track) [4in6]
- RFC 2529, "Transmission of IPv6 over IPv4 Domains without Explicit Tunnels", March 1999 (Standards Track) [6over4]
- **RFC 3056, "Connection of IPv6 Domains via IPv4 Clouds", February 2001 (Standards Track) [6to4]**
- **RFC 3068, "An Anycast Prefix for 6to4 Relay Routers", June 2001 (Standards Track) [6to4]**
- RFC 3964, "Security Considerations for 6to4", December 2004 (Informational) [6to4]
- **RFC 4213, "Basic Transition Mechanisms for IPv6 Hosts and Routers", October 2005 (Standards Track) [Dual Stack, 6in4]**
- **RFC 4380, "Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs)", February 2006 (Standards Track) [Teredo]**
- RFC 5158, "6to4 Reverse DNS Delegation Specification", March 2008 (Informational) [6to4]
- **RFC 5214, "Intra-Site Automatic Tunnel Addressing Protocol (ISATAP)", March 2008 (Informational) [ISATAP]**

- **RFC 5569, "IPv6 Rapid Deployment on IPv4 Infrastructures (6rd)", January 2010 (Informational) [6rd]**
- RFC 5579, "Transmission of IPv4 Packets over Intra-Site Automatic Tunnel Addressing Protocol (ISATAP) Interfaces", February 2010 (Informational)
- RFC 5571, "Softwire Hub and Spoke Deployment Framework with Layer Two Tunneling Protocol Version 2 (L2TPv2)", June 2009 (Proposed Standard) [Softwire]
- **RFC 5572, "IPv6 Tunnel Broker with the Tunnel Setup Protocol (TSP)", February 2010 (Experimental) [TSP]**

For Translation between IPv4 and IPv6, you have to go to the Internet Drafts (submitted documents that are not yet approved and given RFC numbers). This is the cutting edge of IPv6. Drafts have a lifetime of six months, and often go through many iterations before being approved (the iteration number is the last part of the draft name, e.g. -08). To keep track of them, use:

http://datatracker.ietf.org/doc/search

- draft-guo-software-6rd-IPv6-config-00, "IPv6 Host Configuration in 6rd Deployment", 2010-13-01
- draft-howard-isp-ip6rdns-03, "Reverse DNS in IPv6 for Internet Service Providers", 2010-03-08
- draft-ietf-softwire-IPv6-6rd-08, "IPv6 via IPv4 Service Provider Networks '6rd'", 2010-03-23
- draft-lee-softwire-6rd-udp-00, "UDP Encapsulation of 6rd", 2009-10-18
- draft-ietf-behave-v6v4-xlate-stateful-11, "Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers", 2010-03-30
- draft-boucadair-behave-dns64-discovery-00, "DNS64 Service Location and Discovery", 2009-10-18
- draft-cao-behave-dsdns64-00, "Dual Stack Hosts with DNS64", 2010-02-12
- draft-ietf-behave-dns64-09, "DNS64: DNS extensions for Network Address Translation from IPv6 Clients to IPv4 servers", 2010-03-30
- draft-wing-behave-dns64-config-02, "DNS64 Resolvers and Dual-Stack Hosts", 2010-02-12
- draft-templin-isatap-dhcp-06, "Dynamic Host Configuration Protocol (DHCPv4) Option for the Intra-Site Automatic Tunnel Addressing Protocol (ISATAP)", 2009-12-08
- draft-ietf-softwire-dual-stack-lite-04, "Dual-Stack Lite Broadband Deployments Following IPv4 Exhaustion", 2010-03-08

7.2 – Transition Mechanisms

There are four general classes of transition mechanisms to help us get from all-IPv4 through a mixture of IPv4 and IPv6, to eventually all-IPv6:

7.2.1 – Co-existence

Co-existence involves all client and server nodes supporting both IPv4 and IPv6 in their network stacks. The only mechanism in this group is the Dual Stack. This is the most general solution but also involves running essentially two complete networks that share the same infrastructure. It does not double

network traffic, as some administrators fear. Any new connection over IPv6 is typically one less connection over IPv4. Over time, an increasing percentage of the traffic on any network will be IPv6, but the only increase in overall traffic will be from the usual suspects (increasing number of applications, users and/or customers), not from supporting Dual Stack. In fact, at some point you will see the total amount of IPv4 traffic begin to decrease. You may see an increase in incoming customer connections due to the ability to now also accept connections from IPv6 users. When YouTube started accepting connections over IPv6, there was an enormous and almost instant jump in IPv6 traffic on the backbone. Many nodes are ready to begin using IPv6 as soon as content is available, because of automated tunneling. In many cases, the end users might not even have been aware that they were now connecting over IPv6.

There is a recent variant of the dual-stack concept called *Dual-Stack Lite* that uses the basic dual-stack design, but adds in IP-in-IP tunneling and ISP based Network Address Translation to allow an ISP to share precious IPv4 addresses among multiple customers. It is defined in draft-ietf-softwire-dual-stack-lite-04, "Dual-Stack Lite Broadband Deployments Following IPv4 Exhaustion".

7.2.2 – Tunneling

Tunneling involves creating IP-in-IP tunnels with a variety of mechanisms to allow sending IPv6 traffic over existing IPv4 infrastructures by adding an IPv4 packet header to the front of an entire IPv6 packet. This treats the entire IPv6 packet, including IPv6 packet header(s), TCP/UDP header and payload fields as a "black box" payload of an IPv4 packet. In the later phases of the transition, it reverses this: it treats an entire IPv4 packet, including IPv4 packet header and options, TCP/UDP header, and payload fields as a "black box" payload of an IPv6 packet. Some of these tunnel mechanisms are "automatic" (no setup required). Others require manual setup. Some require authentication while others do not. The benefit is to leverage the existing IPv4 infrastructure as a transport for IPv6 traffic, without having to wait for ISPs and equipment vendors to support IPv6 everywhere before anyone can start using it. This allows early adopters to deploy nodes and entire networks today, regardless of whether or not their ISP supports IPv6 today. In some cases (e.g. tunnels to a gateway router or firewall), when the ISP does provide dual-stack service, it is a simple process to change from tunneled service to direct service, and the process is largely transparent to inside users. There are several organizations providing *free* tunneled IPv6 service (using various tunnel mechanisms) during the transition, to help with the adoption of IPv6. Tunneling mechanisms include 6in4, 4in6, 6to4, 6over4, TSP (from gogonet) and Teredo. There are many Operating Systems features and installable client software available to make use of these tunneling mechanisms.

7.2.3 – Translation

This is basically Network Address Translation (with all of its attendant problems), this time between IPv4 and IPv6 (as opposed to the more traditional NAT which is IPv4 to IPv4). An IPv6 to IPv4 translation gateway allows an IPv6-only internal node to access external IPv4-only nodes and allow replies from those legacy IPv4 nodes to be returned to the originating internal IPv6 node. Connections from an internal IPv6-only node to external IPv6-only or dual-stack nodes would be done as usual over IPv6 (without going through the translation gateway). This would be useful for deploying IPv6-only nodes in a predominantly IPv4 world. An IPv4 to IPv6 gateway would allow an IPv4-only internal node to access external IPv6-only nodes, and allow replies from those external IPv6 nodes to be returned to the internal IPv4-ony node. Connections from an internal IPv4-only node to external IPv4-only nodes, or to

dual-stack nodes, would be done as usual over IPv4 (without going through the translation gateway). This would be useful for deploying IPv4-only nodes in a predominantly IPv6 world. Some of these mechanisms require considerable modification to (and interaction with) DNS, such as NAT-PT and NAT64 + DNS64.

There are two broad classes of Network Address Translation between IPv4 and IPv6 – those that work at the IP layer, and are transparent to upper layers and protocols; and those that work at the application layer (i.e. Application Layer Gateways, also called Proxies). The IP layer mechanisms need only be implemented once, for all possible application layer protocols. Unfortunately they also have the most technical issues.

There are quite a few Network Address Translation mechanisms between IPv4 and IPv6 currently proposed in IETF drafts, and all have advantages and disadvantages of various kinds. None is a "clean" design without any problems. One such mechanism called NAT-PT was defined in RFC 2766, "Network Address Translation – Protocol Translation (NAT-PT)", February 2000. It had so many issues that it has already been deprecated to historic status by RFC 4966, "Reasons to Move the Network Address Translator – Protocol Translator (NAT-PT) to Historic Status", July 2007. Reading RFC 4966 gives a lot of insight into the issues with trying to do translation between IPv4 and IPv6 at the Internet Layer.

7.2.4 – Proxies (Application Layer Gateways)

The other kind of translation mechanism takes place at the Application Layer. They are called proxies, because they do things "on behalf of" other servers, much like a stock proxy voter will vote your stock on your behalf. They are also called Application Layer Gateways (ALGs) because they are gateways (they do forwarding of traffic from one interface to another), and they work at the Application Layer of the TCP/IP four-layer model. They don't have the serious problems found in IP layer translation mechanisms, such as dealing with IP addresses embedded in protocols (like SIP or FTP). However, there are some problems unique to proxies:

A proxy must be written for every protocol to be translated, and often even different proxies for incoming and outgoing traffic, even for a given protocol (e.g. "SMTP in" and "SMTP out"). Typically each proxy is a considerable amount of work. Often only a handful of the most important protocols will be handled by proxies, while all other protocols are handled by packet filtering.

Writing a proxy involves implementing most or all of the network protocol, although sometimes in a simplified manner (e.g. there is no need to store incoming E-mail messages in a way suitable for retrieval by POP3 or IMAP, they just need to be queued by destination domain for retransmission by SMTP).

Proxies can support SSL/TLS, but the secure connection extends only from client to proxy, and/or from proxy to server (not directly from client to server). This includes both encryption (the traffic will be in plaintextdstm on the proxy) and authentication (authentication is only from server to proxy, and/or proxy to client, not from server to client). Typically another digital certificate is required for the proxy server if it supports SSL/TLS (in addition to the one for the server).

Proxies can't work with traffic secured in the IP layer (IPsec ESP), without access to the keys necessary to decrypt the packets.

Throughput is typically lower than with a packet filtering firewall, due to the need to process the protocol. Of course the security is much better – it won't let through traffic that is not a valid implementation of the specific protocol, while packet filtering might let through almost anything so long as it uses the right port. There is typically no problem dealing with IP addresses embedded in a protocol. In many cases, the proxies are not transparent, which means the client must know that it is talking not directly to a server, but via an intermediate proxy. Many protocols support this kind of operation, e.g. HTTP provides good support for an HTTP proxy. Basically, there must be a way for a client to specify not only the nodename of the final server, but also the address or nodename of the proxy server. In a browser (HTTP client), the nodename of the final server is specified as usual, and the address of the proxy server is specified during the browser configuration ("use a proxy, which is at address w.x.y.z"). When configured for proxy operation, the browser actually connects to the proxy address and relays the address of the final server to the proxy. The proxy then makes an ongoing connection to the final web server. Some protocols have no support for proxy type operation (e.g. FTP). It is possible for a firewall to recognize outgoing traffic over a given port, and automatically redirect it to a local proxy.

Application Layer Gateways (e.g. for SIP, HTTP, and SMTP) work quite well. Basically they accept a connection on one interface of a gateway, and make a second "ongoing" connection (on behalf of the original node) via another interface of the same gateway. It is easy for the two connections to use different IP versions (e.g. translate IPv4 traffic to IPv6 traffic or vice versa). In some ALGs an entire message might be spooled onto temporary storage, (e.g. email messages) and then retransmitted later. In other cases, the ongoing connection would be simultaneous with the incoming connection, and bidirectional (e.g. with HTTP). This would correspond to a human "simultaneous translator" who hears one language (e.g. Chinese), translates and simultaneously speaks another language (e.g. English).

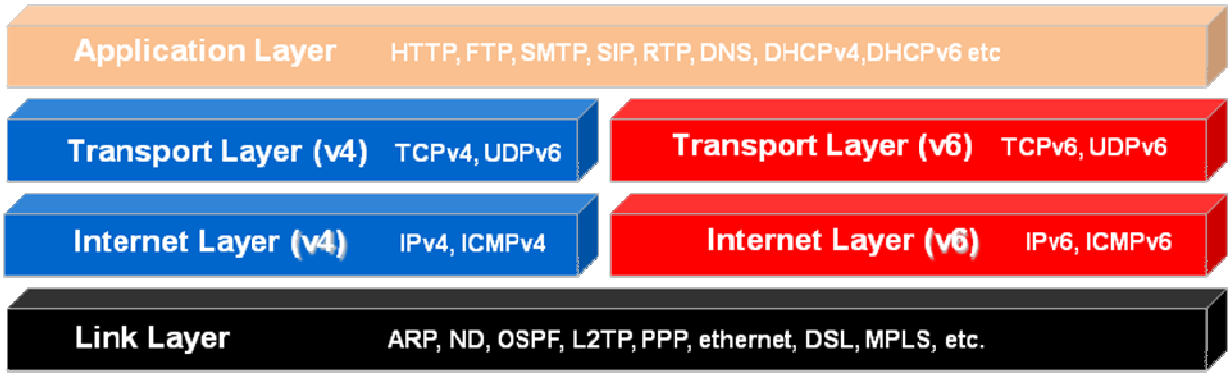
Another example of this is an outgoing web proxy which could accept connections from either IPv4-only or IPv6-only browsers, and then make an ongoing connection to external servers using whatever version of IP those servers support (based on DNS queries). Again, this is a traditional (forward) web proxy, with the addition of IP version translation. This would allow IPv4-only or IPv6-only clients to access any external web server, regardless of IP version they support. Such a proxy could of course also provide any services normally done by an outgoing web proxy, such as caching and URL filtering.

Another example of this is a dual-stack façade that would accept incoming connections from outside over either IPv4 or IPv6, and make an ongoing connection over IPv4 to an internal IPv4-only (or over IPv6 to an IPv6-only) web server. It would relay the web server's responses using whatever version of IP was used in the original incoming connection to the client. This is a typical "reverse" web proxy, with the addition of IP version translation. This kind of translation can help you provide dual stack versions of your web services quickly and easily, without having to dual-stack the actual servers themselves. The same technique could allow you to make your E-mail services dual stack without having to modify your existing mail server.

7.3 – Dual Stack

Dual Stack (also known as *Dual IP Layer*) is defined in RFC 4213, "Basic Transition Mechanisms for IPv6 Hosts and Routers", October 2005. A dual-stack node should include code in the Internet Layer of its network stack to process both IPv4 and IPv6 packets. Typically, there is a single Link Layer that can send and receive either IPv4 or IPv6 packets. The Link Layer also contains both the IPv4 Address Resolution

Protocol (ARP) and the IPv6 Neighbor Discovery (ND) protocol. The Transport Layer has only minor differences in the way IPv4 and IPv6 packets are handled, primarily concerning the way the TCP or UDP checksum is calculated (they checksum also covers the source and destination IP address from the IP header, which of course is different in the two IP versions). The application layer code can make calls to routines in the IPv4 socket API, the IPv6 basic socket API and the IPv6 advanced socket API. IPv4 socket functions will access the IPv4 side of the IP layer, and IPv6 socket functions will access the IPv6 side of the IP layer.



Four Layer Network Model for Dual Stack

The node should include the ability to do conventional IPv4 node configuration (including a node address, default gateway, subnet mask and addresses of DNS servers, all as 32 bit IPv4 addresses). This configuration information can be done manually, via DHCPv4, or some combination thereof. The node should also include the ability to do conventional IPv6 node configuration (including a link-local IP address, one or more global unicast address(es), a default gateway, the subnet length and the addresses of DNS servers, all 128 bit IPv6 addresses). This configuration information can be done manually, automatically via Stateless Address Autoconfiguration, automatically by DHCPv6, or by some combination thereof. There is usually a way to disable either the IPv6 functionality (in which case the node behaves as an IPv4-only node), or the IPv4 functionality (in which case the node behaves as an IPv6-only node). There may or may not also be some tunneling mechanism involved. If the node is in a native dual-stack network no tunnel mechanism is required. If the node is in an IPv4-only, or an IPv6-only network, it will require a tunnel mechanism to handle traffic of the other IP version.

IPv4-only (and for that matter, IPv6-only) applications (client, server and peer-to-peer) will work just fine on a dual-stack node. They will make calls to system functions only one side of the network stack. They will not gain any new ability to accept or make connections over the other IP version just because they are running on a dual-stack node.

A dual-stack client can connect to IPv4-only servers, IPv6-only servers, or dual-stack servers. A dual-stack server can accept connections from IPv4-only clients, IPv6-only clients, or dual-stack clients. Dual-stack is the most complete and flexible solution. The only issues are the additional complexity of implementation and deployment, and the additional memory requirements. For very small devices (typically clients), dual-stack may not be an option. Some critics of IPv6 claim that Dual Stack is not

viable because we are running out of IPv4 addresses. What they are missing is that there are *plenty* of private IPv4 addresses for use behind NAT, and the IPv4 side of dual-stack systems can be used only for protocols where this is not a problem, while using their IPv6 side for those protocols which are incompatible with NAT (IPsec VPN, SIP, Mobile IP, P2P, etc.), or can benefit from other IPv6 features which are superior to their IPv4 equivalents, such as multicast and QoS (for SIP, IPTV, conferencing, P2P, etc.). Also, any application running on that node that needs to accept a connection from external nodes (e.g. your own web server) can use a global unicast IPv6 address (for IPv6 capable clients). If you want to accept connections from IPv4 clients, you would have needed a globally routable IPv4 address for that anyway, or would need to deploy NAT Traversal (with or without dual-stack). Dual-Stack cannot create more globally routable IPv4 addresses. It can however, allow you to easily make use of an almost unlimited number of globally routable IPv6 addresses (both unicast and multicast).

A key part of a dual-stack network is a correctly configured dual-stack DNS service. It should not only be able to publish both A and AAAA records (as well as reverse PTR records for IPv4 and IPv6), it should also be able to accept queries and do zone transfers over both IPv4 and IPv6. A dual-stack network typically uses DHCPv4 to assign IPv4 addresses to each node, and either Stateless Address Autoconfiguration and/or DHCPv6 to assign IPv6 addresses to each node. A dual-stack firewall can bring in either direct dual-stack service (both IPv4 and IPv6 traffic) from an ISP (if available), routing both to the inside network; or it can bring in direct IPv4 traffic from an ISP and terminate tunneled IPv6 traffic (from a "virtual" ISP usually different from the IPv4 ISP) and route both IPv4 and IPv6 into the inside network. In either case (direct dual stack service or tunneled IPv6 with endpoint in the gateway), inside nodes appear to have direct dual-stack service, and require no support for tunneling.

The DNS support does not require any modifications to a standard DNS server (e.g. BIND). DNS just needs to be able to perform its normal forward and reverse lookups with either IPv4 (A/PTR) or IPv6 (AAAA/PTR) resource records. There is need to for the DNS server to do nonstandard mappings between IPv4 and IPv6 addresses as is required with most IP layer translation schemes (e.g. NAT64 + DNS64).

Migrating IPv4-only client or server applications to IPv6-only is quite simple. There is essentially a one to one mapping of function calls from the IPv4 socket API to similar ones in the IPv6 basic socket API. Of course, more storage is required for each IP address in data structures (4 bytes for IPv4 addresses, 16 bytes for IPv6 addresses).

Modifying either IPv4-only clients or IPv4-only servers to *dual-stack* operation is somewhat more complicated. A dual-stack client must be modified to retrieve multiple addresses (IPv4 and/or IPv6) from DNS, and try connections sequentially to the returned address list until a connection is accepted. The default (assuming IPv6 connectivity is available) is to attempt connections over IPv6 first. If DNS advertises an IPv6 address, and the node supports IPv6, but for some reason the client is unable to connect over IPv6 (e.g. tunnel down), there will be a 30 second timeout then a fallback to IPv4. A dual-stack server must listen for connections on *either* IPv4 *or* IPv6, and process connections from either. It is also possible to deploy two copies of each server, one being IPv4-only, and the other IPv6-only. This might involve cross process file locking on any shared resource, such as a message store. Either approach to providing dual-stack servers will work fine, and the user experience will be the same. Conditional compilation could be used to have a single source code tree create both an IPv4-only and an IPv6-only executable (depending on setting of system variables at compilation time). For most server designs (process per connection or thread per connection), the split model (an IPv4-only server and an IPv6-only server) would roughly double the memory footprint compared to a single dual-stack server.

Most open source servers today have good support for dual-stack operation. These include the Apache web server, Postfix SMTP server, Dovecot IMAP/POP3 mail access servers, etc. If you are a developer and want to see examples of how to deploy dual-stack servers, there are numerous examples available in open source. Most open source client software also has good support for IPv6 and dual-stack. These include the Firefox web browser, Thunderbird E-mail client, etc. The open source community has done an excellent job of supporting the migration to IPv6. Both the original IPv4-only socket API and the newer IPv6 socket APIs are readily available on all UNIX and UNIX-like platforms. The documentation for the newer IPv6 socket APIs are in RFC 3493, "Basic Socket Interface Extensions for IPv6" and RFC 3542, "Advanced Sockets Application Program Interface (API) for IPv6". There is also RFC 5014, "IPv6 Socket API for Source Address Selection", and RFC 4584, "Extension to Sockets API for Mobile IPv6".

Virtually all Microsoft server products (since 2007) have had good support for dual-stack operation. These include Windows Server 2008 (and all of its components, such as DNS, file and printer sharing, etc.); Exchange Server 2007; and many others. Their client operating systems have had good support for IPv6 since Vista. For Microsoft developers, both the original IPv4-only socket API (WinSock) and the new IPv6 socket APIs (basic and advanced) are available as part of the standard Microsoft developer libraries.

7.3.1 – Dual-Stack Lite

The IETF Softwires Working Group has come up with a variant on the basic Dual-Stack network design, which is described in draft-ietf-softwire-dual-stack-lite-04, "Dual-Stack Lite Broadband Deployments Following IPv4 Exhaustion". Clients will still support both IPv4 and IPv6, but the service from the ISP to the customer will be IPv6-only, with IPv4 service tunneled over the IPv6. The addresses provided to the customer will be RFC 1918 *private addresses*, provided by a giant Large Scale NAT (LSN) at the ISP. The NAT involved actually uses the customer's IPv6 address to tag the private IPv4 addresses used by the client, which would allow multiple ISP clients to use the same private address range (e.g. all of them could use 10.0.0.0/8, and the LSN would keep each organization's addresses separate based on their unique IPv6 address).

IPv6-only or Dual-Stack nodes at the client would be able to connect to any IPv6 node in the world directly, via the ISPs IPv6 service. IPv4-only or Dual-Stack nodes at the client would be able to connect to any IPv4 node in the outside world via IPv4 tunneled over IPv6, with addresses from the ISP's Large Scale NAT. There is no 6to4 translation that would allow an IPv6-only node to connect to external IPv4 nodes, or 4to6 translation that would allow an IPv4-only node to connect to external IPv6 nodes. Any internal node that needs to connect to external IPv4 nodes should be configured to support dual-stack.

The way this differs from basic Dual-Stack operation is that there is no direct IPv4 service provided, and the IPv4 addresses used at the client are private, and managed by infrastructure at the ISP. This allows the ISP to share a relatively small number of precious real IPv4 addresses among a large number of customers, and also allows the ISP to run only a single IP family of "direct service" to the customer (which happens to be IPv6). A major advantage of DS-Lite is that no 6to4 or 4to6 translation is required.

This will require a firmware upgrade (or replacement) of the Customer Premise Equipment (CPE), which is typically a DSL or cable modem, with embedded router and NAT.

This is the “IPv6 service” being trialed by Comcast in the United States as this is being written.

The ISC (who also supply the BIND DNS server and dhcpd DHCPv4 server) have created a freeware implementation of the ISP side facilities to support DS-Lite, called AFTR (Address Family Transition Router). This includes IPv4 over IPv6 tunneling, DHCPv4, DHCPv6 and some other pieces.

The CPE device for DS-Lite is called B4 (Basic Bridging BroadBand Element). There is an open source implementation of this for the Linksys WRT-54GL.

7.4 – Tunneling

If tunneled service is brought into the network by a gateway device (typically the gateway router or firewall), which contains the tunnel endpoint, the internal network is a native dual-stack network from the viewpoint of all internal nodes. No internal node needs to have support for any tunneling mechanism. If at some point, the tunneled service is replaced with direct service (both IPv4 and IPv6 service direct from your ISP) a minor reconfiguration at the gateway is all that is required. Internal nodes will probably not require any reconfiguration at all. They will typically have a new IPv6 prefix (unless you were getting tunneled service from your ISP), so you will likely have to update all forward and reverse address references in your DNS server (only for IPv6 addresses), to reflect the new IPv6 prefix. If your DNS server supports *instant prefix renumbering* like SolidDNS, this is a quick, painless process. If you are using DHCPv6 in stateful mode (where it assigns IP addresses) in conjunction with dynamic DNS registration, even DNS changes due to change of IPv6 prefix may happen automatically.

A tunnel mechanism has both a server side and a client side. The server side typically can accept one or more connections from tunnel clients. It is also commonly called a *Tunnel Broker*. A tunnel client typically makes connections to a single tunnel server. Some such connections (e.g. with 6in4) are not authenticated (although the server can typically be restricted to accepting connections only from specific IP addresses or address ranges). Some such connections (e.g. gogonet’s TSP) include authentication of the client to the server, before the tunnel will begin operation. Some connections (e.g. 6in4) require a globally routable IPv4 address on the client (although this can be the same address as the hide-mode NAT address). Other tunnel clients will work behind NAT, even with a private address. These include a NAT traversal mechanism in the client, and typically all tunneled packets are carried over UDP. Once a tunnel is created, it is bi-directional. Packets can be sent either upstream or downstream. From a hop count perspective, the tunnel counts as one hop, no matter how many hops the tunneled packets traverse.

Typical Product Support for Tunneling: InfoWeapons’ SolidWall Dual Stack Firewall

As an example of a typical product that includes support for tunneling, InfoWeapons has a dual stack firewall called SolidWall. On the IPv4 side, it includes typical firewall capabilities including routing, filtering by port and address, stateful packet inspection and various forms of NAPT (hide mode, BINAT or 1:1 and port forwarding). On the IPv6 side, it includes all of that (except for NAPT), plus a Router Advertisement Daemon (to enable Stateless Address Autoconfiguration) and 6in4 server and client mode. You could use the 6in4 client mode to bring in IPv6 tunneled service from any 6in4 virtual ISP (e.g. Hurricane Electric). You could create your own (small) IPv6 virtual ISP using Solidwall’s 6in4 tunnel server mode. For example, you could provide tunneled IPv6 service from your HQ or collocation facility

to various branches, using the 6in4 tunnel server at HQ and the 6in4 tunnel clients at each branch. You can carve off any number of "/64" subnets into each branch office. For example, you could split a "/48" block into sixteen "/52" blocks, and route one "/52" block into each branch office.

Because the client mode tunnel endpoint is located inside a firewall node, incoming IPv6 packets from the tunnel can be filtered and routed into any inside network(s). Outgoing IPv6 packets from any internal network can be filtered and routed out the tunnel to the outside world (via a remote tunnel server).

The server mode tunnel endpoint is also located inside a firewall node, so the firewall's routing capabilities allow you to easily route any block of addresses from the outside world into any tunnel (and hence to branch offices), and outgoing packets (from tunnels coming from branch offices) to the outside world. Currently there is no support for OSPFv3 or BGP4+, so you would need to relay outgoing IPv6 traffic onwards via an ISP (or virtual ISP) that can do further routing.

As an example, we have one SolidWall at our colocation facility in Atlanta, Georgia. We split our "/48" block (*2001:418:5403::/48*) into 16 "/52" blocks. We use subnet *2001:418:5403:0::/52* locally in the collocation facility. Our website (www.infoweapons.com) is hosted there along with our main DNS servers (on a pair of our own SolidDNS appliances). We route subnet *2001:418:5403:2000::/52* to our office in Cebu, and subnet *2001:418:5403:3000::/52* to my home in Cebu. The client side of each of these tunnels is another SolidWall appliance. In each case, it brings IPv4 service from a local ISP into the network, with IPv6 from our colo tunneled over it. Incoming IPv6 is "de-tunneled" inside the firewall and routed into the internal network alongside the filtered IPv4. Outgoing IPv6 from each network is "en-tunneled" through the local SolidWall, and it comes out at the SolidWall at the colo, where it is routed onwards via our direct dual-stack service from NTT America. It sounds complicated, but in reality it only takes a few minutes to each tunnel up, using a simple point-and-click GUI.

Because the tunnel mechanism used (6in4) is an IETF standard, SolidWall's tunnels will interoperate with server or client mode 6in4 tunnel endpoints on any other vendor's products or even on homegrown routers built from open source platforms such as FreeBSD, OpenBSD or Linux.

In future releases we plan to add several other kinds of tunneling to SolidWall (probably 6to4, TSP and Dual-Stack Lite).

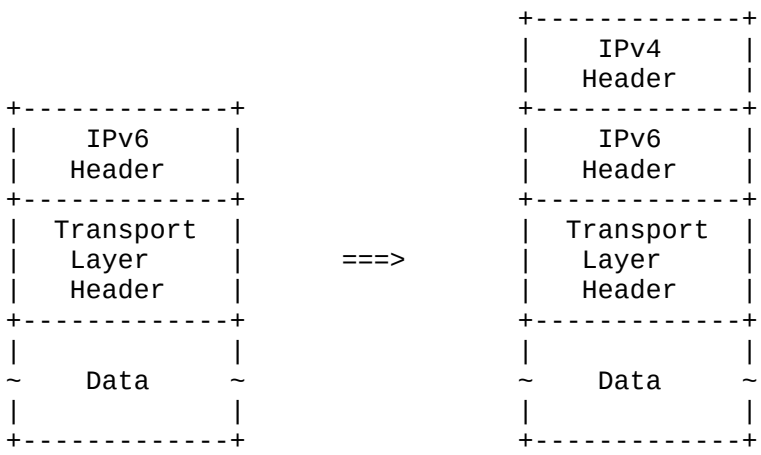
7.4.1 – 6in4 Tunneling

RFC 4213 (in addition to specification for Dual Stack), specifies 6in4 tunneling (unfortunately they use the term "6over4" when they mean "6in4"). Technically, *6in4* is a *tunneling* mechanism. *6over4* is a *transition* mechanism that uses 6in4 tunneling to create a virtual IPv6 link over an IPv4 multicast infrastructure (see RFC 2529). This book will use the term *6in4* unless we are specifically talking about 6in4 tunnels over IPv4 multicast. 6in4 is also sometimes referred to as "Protocol 41" tunneling. 6in4 tunneling requires both ends of the tunnel to have globally routable IPv4 addresses (neither tunnel endpoint can be behind NAT). It is possible for a firewall that is using a globally routable IPv4 address for HIDE mode NAT (with multiple internal nodes hidden behind it), to use that same address as one endpoint of a 6in4 tunnel.

6in4 Encapsulation

This process is done to “push packets into the tunnel” for packets going from either end of the tunnel to the other. The basic idea is to prepend a new IPv4 packet header to a complete IPv6 packet (which itself consists of the basic IPv6 header, zero or more extension headers, a TCP or UDP header, and a payload), and treat the entire IPv6 packet as a “black box” payload for the IPv4 packet.

The encapsulation of an IPv6 datagram in IPv4 for 6in4 tunneling is shown below:



Encapsulating IPv6 in IPv4

The new IPv4 packet header is constructed as follows (from the RFC):

IP version:

4 (the encapsulating packet is IPv4)

IP Header Length:

5 (in 32 bit words, so 20 bytes, and no IPv4 options are used in the encapsulating header)

Type of Service:

0 unless otherwise specified (see RFC 2983 and RFC 3168 for details)

Total Length:

IPv6 payload length plus IPv6 header length (40) plus IPv4 header length (20), so IPv6 payload length + 60

Identification:

Generated uniquely as for any IPv4 packet

Flags:

DF (Don't Fragment) flag should be set as specified in section 3.2 of RFC 4213
MF (More Fragments) flag set as necessary if fragmenting

Fragment Offset:

Set as necessary if fragmenting

Time to Live (TTL):

Set as described in Section 3.3 of RFC 4213

Protocol:

41 – this is the defined payload type for IPv6 tunneled over IPv4, and is used regardless of whether the IPv6 transport is UDP or TCP

Header Checksum:

Calculated as usual for an IPv4 packet header

Source Address:

An IPv4 address of the encapsulator: either configured by the administrator, or an address of the outgoing interface

Destination Address:

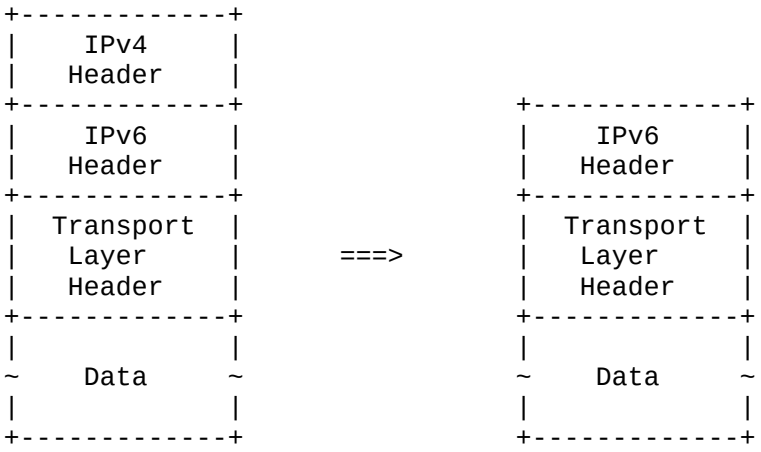
IPv4 address of the tunnel endpoint (that is, the client side of the tunnel)

6in4 Decapsulation

This is done for all packets received over the tunnel from the other end. The basic idea is to strip the outer (IPv4) packet header off (and discard it), then handle what is left (the original IPv6 packet) as native IPv6 traffic.

From the RFC: When a dual stack node receives an IPv4 datagram that is addressed to one of its own IPv4 addresses (or a joined multicast group address), which has a protocol field of 41 (tunneled IPv6), the packet must be verified to belong to a configured tunnel interface (according to source/destination addresses), reassembled (if it was fragmented), have the IPv4 header removed, and then the resulting IPv6 datagram is submitted to the IPv6 layer on the node for further processing.

The decapsulation process for 6over4 tunneling is shown below:



Decapsulating IPv6 from IPv4

According to RFC 4213, section 3.2, the MTU of the tunnel *must* be between 1280 and 1480 bytes (inclusive), but *should* be 1280 bytes. Section 3.3 specifies that the tunnel counts as a *single* hop to IPv6, regardless of how many hops the underlying IPv4 packet traverses. The actual TTL value in the outer IP header should be set as for any IPv4 packet (see RFC 3232 and RFC 4087).

RFC 4213 section 3.4 specifies how to handle errors that happen while the encapsulated packet is *inside* the tunnel. Unfortunately older routers may not return enough of the packet to include both source and destination IPv6 addresses of the encapsulated packet, so it may not be possible to construct a correct ICMPv6 error message. Newer routers typically include enough of the failed packet for correct ICMPv6 error message creation.

7.4.2 – 6over4 Tunneling

6over4 tunneling is defined in RFC 2529, “Transmission of IPv6 over IPv4 Domains without Explicit Tunnels”. It is a transition mechanism that uses 6in4 tunneling over an IPv4 multicast capable network. The term 6over4 is sometimes confusingly used for 6in4 tunneling. Due to the requirement for IPv4 multicast, which is very difficult to deploy, 6over4 is not commonly used.

7.4.3 – 6to4 Tunneling

6to4 tunneling is described in RFCs 3056, 3068, 3964 and 5158. It is a transition mechanism that provides tunneled IPv6 over IPv4 without explicitly configured tunnels. With the original 6to4 mechanism, the IPv4 addresses involved must be valid globally routable IPv4 addresses (not behind NAT). Teredo is a variant of 6to4 tunneling that will work even behind NAT.

6to4 does not provide general translation to IPv4 addresses for interoperation between IPv6 hosts and IPv4 hosts. It uses automatically created tunnels over IPv4 to facilitate communication between any number of IPv6 hosts.

A “6to4 Host” is a regular IPv6 host that also has at least one 6to4 address assigned to it.

A “6to4 Router” is a regular IPv6 router that includes a 6to4 pseudo interface. It is normally a border router between an IPv6 site and a wide-area IPv4 network.

A “6to4 Relay Router”, which is a 6to4 capable router which is also configured to support transit routing between 6to4 addresses and native IPv6 addresses.

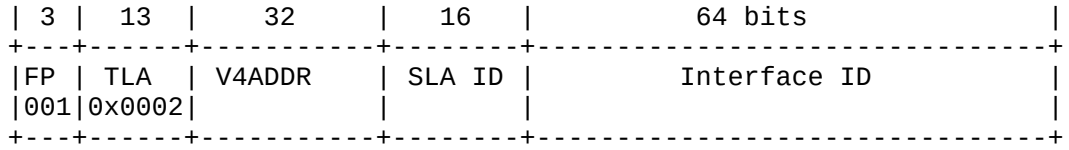
Without 6to4 Relay Routers, you can communicate with other nodes that use 6to4 tunneling, over IPv6 (even though your ISP does not yet support IPv6). To communicate with IPv6 users that are *not* using 6to4, you need to relay your traffic through a 6to4 Relay Router. You can create your own Relay Router. It must have both a 6to4 pseudo interface, and native (not 6to4) IPv6 connectivity to the IPv6 Internet.

A 6to4 router will send an encapsulated packet directly over IPv4 if the first 16 bits of an IPv6 destination address are 2002, using the next 32 bits as the IPv4 destination (which must be another 6to4 node that will unpack the IPv6 packet being sent and use it or relay it to other IPv6 hosts). For all other IPv6 destination addresses, a 6to4 router will forward the packet to the IPv6 address of a well-known Relay Router that has access to native IPv6 (or simply send it to the IPv6 anycast address 2002:c058:6301::/128, which will send it to the nearest available 6to4 Relay Router).

For details on how to configure a FreeBSD node with 6to4 tunneling, see

> http://www.kfu.com/~nsayer/6to4

An IPv6 address for use with 6to4 tunneling looks like the following:



Essentially the IPv6 prefix for all 6to4 addresses is 2002:V4ADDR::/48.

RFC 2374 defines SLA ID as follows:

> “The SLA ID field is for a Site Level Aggregator Identifier. This can be used by individual organizations to create its own local addressing hierarchy and to identify subnets. It is analogous to subnets in IPv4, except that each organization has a much greater number of subnets.”

RFC 3056 defines a 6to4 pseudo-interface as follows:

> “6to4 encapsulation of IPv6 packets inside IPv4 packets occurs at a point that is locally equivalent to an IPv6 interface, with the link layer being the IPv4 unicast network. This point is referred to as the pseudo-interface. Some implementers may treat it exactly like any other interface, and others may treat it like a tunnel endpoint.”

7.4.4 – Teredo

Teredo is one extension of basic 6to4 tunneling. It adds encapsulation over UDP datagrams, and uses a simplified version of STUN NAT Traversal, allowing a Teredo client to be behind NAT. It is defined in RFC 4380, "Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs)". The name "Teredo" is part of the Latin name for a little worm that bores holes through wooden ship hulls. This gives you a pretty good idea of what the Teredo protocol does to your firewall. Teredo is installed and enabled by default in Windows Vista and Windows 7. It is possible to disable it, but this is not easy.

There is an open source Teredo client for Linux, BSD and Mac OS X called *Miredo*. It can act as a client, relay and server.

There are publically available Teredo "relay routers" that allow any node with Teredo to access the IPv6 Internet. Microsoft makes several very large ones available for use from Windows nodes. Windows nodes are preconfigured to use these relay servers. Unlike 6to4 and some other tunnel mechanisms, Teredo can only provide a single "/128" IPv6 address per tunnel endpoint.

Teredo uses a different IPv6 address block than basic 6to4 tunneling. The rest of the Teredo address is defined differently as well.

- Bits 0-31 contain the Teredo prefix, which is 2001:0000::/32.
- Bits 32-63 contain the IPv4 address of the Teredo Server used.
- Bits 64-79 contain some flags. Currently only bit 64 is used. If set to 1, the client is behind a cone NAT, otherwise it is 0. More of these flag bits are used in Vista, Windows 7 and Windows Server 2008.
- Bits 80-95 contain the obfuscated UDP port number (port number that is mapped by NAT, with all bits inverted).
- Bits 96-127 contain the obfuscated IPv4 address of the node (public IPv4 address of the NAT with all bits inverted).

As an example, a Teredo address might be 2001::4136:e378:8000:63bf:3fff:fdd2, which broken into fields is:

- Bits 0-31: 2001:0000 – The Teredo prefix
- Bits 32-63: 4136:e378 – IPv4 address 65.54.227.120 in hexadecimal
- Bits 64-79: 8000 – Cone mode NAT
- Bits 80-95: 63bf – Obfuscated port number 40000
- Bits 96-127: 3fff:fdd2 – Obfuscated public IPv4 address of the node (192.0.2.45)

Public Teredo servers can be located on the following website:

http://www.bgpmon.net/teredo.php

Hurricane Electric, as of Q1 2009, had deployed 14 public Teredo relays (via anycast), in Seattle Washington, Fremont California, Los Angeles California, Chicago Illinois, Dallas Texas, Toronto Ontario,

New York New York, Ashburn Virgina, Miami Florida, London England, Paris France, Amsterdam Netherlands, Frankfurt Germany and Hong Kong SAR.

7.4.5 – 6rd – IPv6 Rapid Deployment

6rd is another extension of 6to4 tunneling, that adds reliable routing. Normal 6to4 tunnels use the standard 2002://16 prefix, and in theory scale to the entire world. Unfortunately, there is no way to control who can connect to 6to4 public servers, and there is no incentive to provide quality service. Also there is no guarantee that any 6to4 node will be reachable. The same is true of Teredo.

6rd instead works only within the confines of a single ISP, and instead of the 2000://16 prefix, each ISP uses a prefix that they own and control and runs the Relay Router. They can insure quality service and reachability of all nodes within their network.

6rd was deployed by a French ISP called “Free” (in spite of the name, this is a commercial ISP). This was done in 5 weeks starting in December 2007. This gave France the second highest IPv6 penetration in the world, 95% of which was via Free’s 6rd. RFC 5569 discusses Free’s 6rd deployment. The current Internet Draft that defines 6rd (draft-ietf-softwire-IPv6-6rd-08, “IPv6 via IPv4 Service Provider Networks ‘6rd’”, 2010-03-23) should be approved soon. Meanwhile you can read the draft.

In January 2010, Comcast (a large U.S. ISP) announced plans to do a trial deployment of IPv6 using 6rd. Softbank (a large Japanese ISP) also has announced that they will roll out IPv6 using 6rd.

7.4.6 – Intra-Site Automatic Tunnel Addressing Protocol (ISATAP)

ISATAP is a transition mechanism that allows transmission of IPv6 packets between dual-stack nodes on top of an IPv4 network. It is similar to 6over4, but it uses IPv4 as a virtual Non-Broadcast Multiple-Access (NBMA) network Link Layer, and does not require IPv4 multicast (which 6over4 does require). It is discussed in RFC 5214, “Intra-Site Automatic Tunnel Addressing Protocol (ISATAP)”.

ISATAP specifies a way to generate a link-local IPv6 address from an IPv4 address, plus a mechanism for performing Neighbor Discovery on top of IPv4.

The generated link local address is created by appending the 32 bit IPv4 address onto the 96 bit prefix *fe80:0:0:0:0:5efe*::. For example, the IPv4 address *192.0.2.143* in hexadecimal is *c000028f*. Therefore the corresponding ISATAP link-local address is *fe80::5efe:c000:28f*.

The Link Layer address for ISATAP is not a MAC address, but an IPv4 address (remember IPv4 is used as a virtual Link Layer). Since the IPv4 address is just the low 32 bits of the ISATAP address, mapping onto the “Link Layer” address simply involves extracting the low 32 bits (ND is not required). However, router discovery is more difficult without multicast. ISATAP hosts are configured with a *potential routers list* (PRL). Each of the routers on this list is probed by an ICMPv6 Router Discovery message, to determine which of them are functioning and to then obtain the list of on-link IPv6 prefixes that can be used to create global unicast IPv6 addresses.

Current implementations create their PRL by querying the DNS. DHCPv4 is used to determine the local domain, then a DNS query is done for isatap.<localdomainame>. For example, if the local domain is *demo.com*, it would do a DNS query for *isatap.demo.com*.

ISATAP avoids circular references by only querying DNS over IPv4, but it is still a lower-layer protocol that is using a higher-layer function (DNS). This is a violation of network design principles.

ISATAP is implemented in Windows XP, Windows Vista, Windows 7, Windows Mobile, and Linux (since Kernel 2.6.25). It is not currently implemented in *BSD due to a potential patent issue.

7.4.7 – Tunnel Setup Protocol (TSP)

TSP is defined in RFC 5572, and was created by Marc Blanchet, author of "Migrating to IPv6" (and at the time CTO of Hexago). Hexago sold an IPv6 Tunnel Broker appliance that implemented TSP. It is similar in certain respects to 6in4 tunneling, but works even behind NAT, and includes client authentication. Hexago offered free TSP clients for most platforms and free IPv6 tunneled service to anyone through their own Freenet6 service. There was a management buyout a few years ago, and now Hexago is known as gogo6. They still offer the same TSP based products and free tunneled IPv6 service. They now run gogoNET, which is a social networking site built around IPv6 (with 17,000 members as of April 2010).

gogo6's free tunneled service is one of the best ways to get started in IPv6. You simply download their free client for whatever platform you use, install it, sign up for a free account, and connect. You can obtain a single ("/128") IPv6 address even when you are on the road (e.g. in a hotel). You can even obtain a "/56" block on a router to provide IPv6 connectivity to an entire network. Service is on a "best effort" basis (no guarantees). They also have a DS-Lite trial service for tunneled IPv4 over IPv6, and a DSTM (Dual-Stack Transition Method) service to allow people to access IPv4-only Internet services from an IPv6-only network. The IPv4 address obtained in private, and all users of Freenet6's DSTM share a single block of public IPv4 addresses.

TSP is actually a framework that can do any of the following:

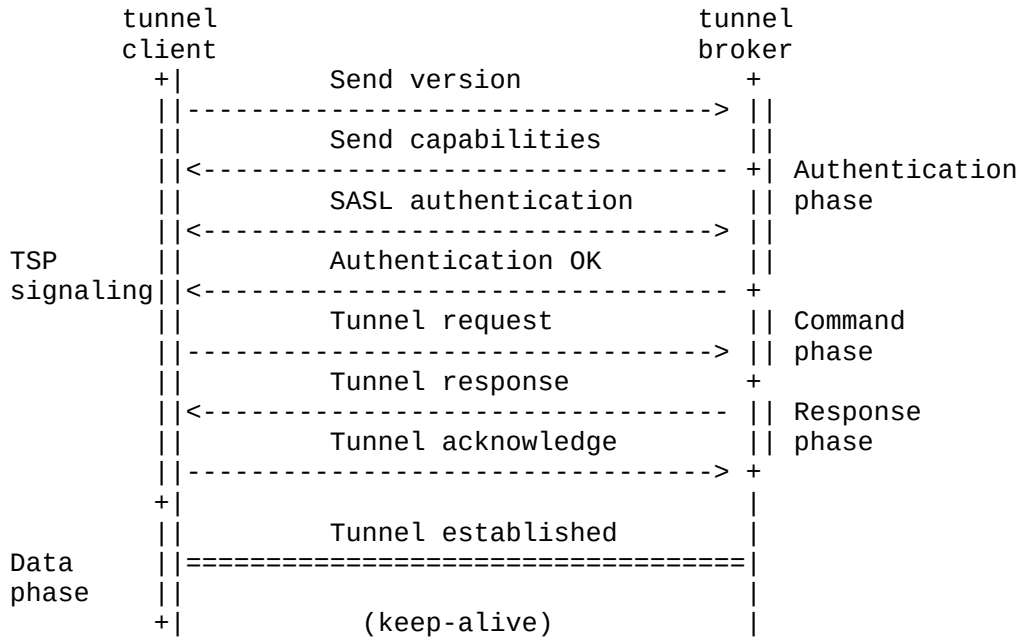
- User authentication, with various mechanisms include Simple Authentication and Security Layer (SASL)
- NAT discovery
- Encapsulation of IPv6 over IPv4 (RFC 4213, 6in4)
- Encapsulation of IPv4 over IPv6 (RFC 2473, 4in6)
- Encapsulation of IPv6 over UDP-IPv4 tunnels for NAT traversal
- IP address assignment for the tunnel endpoints
- DNS registration of the IP endpoint address (AAAA record)

Once a user is authenticated, the client performs NAT discovery. The client sends a TSP message over UDP, containing its tunnel request information (e.g. source IPv4 address) to the tunnel server. The TSP server compares the IPv4 source address of this datagram with the address in the datagram payload. If they differ, then at least one NAT was traversed, and the tunnel will use encapsulation over UDP with NAT traversal. A keep-alive mechanism is also selected. If the addresses match (no NAT detected), then

simple 6in4 (RFC 4213) tunneling is selected. If the datagram contains an IPv6 address, and the source address of the datagram is IPv6, then 4in6 tunneling is selected.

Features of TSP:

- The created tunnel is static, which is more secure than automated tunnels (per RFC 3964).
- The IPv6 address and prefix are stable, and not dependent on the underlying IPv4 address.
- Prefix assignment is supported.
- Automated NAT discovery and negotiation of optimal NAT traversal scheme.
- TSP uses TCPv4 on port 3653 or UDPv4 on port 3653.



Tunnel Setup Protocol Exchange

```
C:VERSION=2.0.0 CR LF
S:CAPABILITY TUNNEL=V6V4 TUNNEL=V6UDPV4 AUTH=ANONYMOUS AUTH=PLAIN
  AUTH=DIGEST-MD5 CR LF
C:AUTHENTICATE ANONYMOUS CR LF
S:200 Success CR LF
```

Successful Anonymous Authentication

```
C:VERSION=2.0.0 CR LF
S:CAPABILITY TUNNEL=V6V4 TUNNEL=V6UDPV4 AUTH=ANONYMOUS AUTH=PLAIN
  AUTH=DIGEST-MD5 CR LF
C:AUTHENTICATE DIGEST-MD5 CR LF
S:cmVhbG09aGV4b3Msbm9uY2U9MTExMzkwODk2OCxxb3A9YXV0aCxhbGdvcml0aG09bWQ
  1LXNlc3MsY2hhcnNldD11dGY4
```

```
C:Y2hhcnNldD11dGY4LHVzZXJuYW1lPSJ1c2VybmFtZTEiLHJlYWxtPSJoZXhvcyIsbm9
  uY2U9IjExMTM5MDg5NjgiLG5jPTAwMDAwMDAxLGNub25jZT0iMTExMzkyMzMxMSIsZG
  lnZXN0LXVyaT0idHNwL2hleG9zIixyZXNwb25zZT1mOGU0MmIzYzUwYzU5NzcxODUzZ
  jYyNzRmY2ZmZDFjYSxxb3A9YXV0aA==
S:cnNwYXV0aD03MGQ1Y2FiYzkyMzU1NjhiZTM4MGJhMmM5MDczODFmZQ==
S:200 Success CR LF
```

Successful Digest-MD5 Authentication

```
-- UDP session established --
C:VERSION=2.0.0 CR LF
S:CAPABILITY TUNNEL=V6V4 TUNNEL=V6UDPV4 AUTH=ANONYMOUS
  AUTH=PLAIN AUTH=DIGEST-MD5 CR LF
C:AUTHENTICATE ANONYMOUS CR LF
S:200 Success CR LF

C:Content-length: 205 CR LF
<tunnel action="create" type="v6udpv4">
 <client>
  <address type="IPv4">192.0.2.135</address>
<keepalive interval="30"></keepalive>
</client>
</tunnel> CR LF

S:Content-length: 501 CR LF
200 Success CR LF
<tunnel action="info" type="v6udpv4" lifetime="604800">
  <server>
    <address type="IPv4">192.0.2.115</address>
    <address type="IPv6">
    2001:db8:8000:0000:0000:0000:0000:38b2
    </address>
  </server>
  <client>
    <address type="IPv4">192.0.2.135</address>
    <address type="IPv6">
    2001:db8:8000:0000:0000:0000:0000:38b3
    </address>
    <keepalive interval="30">
      <address type="IPv6">
      2001:db8:8000:0000:0000:0000:0000:38b2
      </address>
    </keepalive>
  </client>
</tunnel> CR LF

C:Content-length: 35 CR LF
<tunnel action="accept"></tunnel> CR LF
```

Example of a Command/Response Sequence

```
-- Successful TCP Connection --
 C:VERSION=1.0 CR LF
 S:CAPABILITY TUNNEL=V4V6 AUTH=DIGEST-MD5 AUTH=ANONYMOUS
```

```
  CR LF
C:AUTHENTICATE ANONYMOUS CR LF
S:OK Authentication successful CR LF
C:Content-length: 228 CR LF
  <tunnel action="create" type="v4v6">
     <client>
         <address type="IPv6">
         2001:db8:0c18:ffff:0000:0000:0000:0001
         </address>
     </client>
  </tunnel> CR LF
S: Content-length: 370 CR LF
   200 OK CR LF
   <tunnel action="info" type="v4v6" lifetime="1440">
     <server>
        <address type="IPv4" length="30">
        192.0.2.2
        </address>
        <address type="IPv6">
        2001:db8:c18:ffff:0000:0000:0000:0002
        </address>
     </server>
     <client>
        <address type="IPv4" length="30">
        192.0.2.1
        </address>
        <address type="IPv6">
        2001:db8:c18:ffff::0000:0000:0000:0001
        </address>
     </client>
   </tunnel> CR LF
```

IPv4 over IPv6 Tunnel Request and Response

7.4.8 – Softwires

The IETF has a very active *Softwires* working group. Essentially they are trying to create standards for tunneling IPv6 over IPv4 networks and for tunneling IPv4 over IPv6 networks. There are two basic models for this, one is called *Hub and Spoke*. This is similar to the way that airlines have a few large *hub* airports and many *spokes* or local flights radiating from those hubs to smaller communities nearby. For example, Atlanta International Airport is a hub for the entire Southeastern United States. If you fly in or out of that region, you will likely interchange in Atlanta. There are several schemes that vary in exactly what part of the network path the softwire is deployed:

- from ISP to Customer Modem/Router
- from ISP via Customer Modem/Router to an inside softwire router
- from ISP via Customer Modem/Router to an end-user node

All the components necessary to deploy the various schemes are widely available, including;

- LNS: large ISP based *L2TP Network Server*

- Dual AF CPE: Customer Premise Equipment Modem/Router with support for L2TPv2 softwires
- Dual AF Router: Customer Premise dual-stack router with support for L2TPv2 softwires
- Dual AF Host: Client software for end-user nodes with support for L2TPv2 softwires

In the above, "Dual AF" means Dual *Address Family*, in other words, IPv4 + IPv6, or dual stack.

The other softwire architecture is called *Mesh*. This involves a number of peer nodes, with multiple connections between them. If all nodes are connected to all other nodes, that would be a fully meshed network.

The term *softwire* refers to a tunneled link between two or more nodes. In early RFCs related to this technology, sometimes the term *pseudowire* is used instead. Softwires are assumed to be long-lived, and the setup time is expected to be a very small fraction of the total time required for the startup of the Customer Premise Equipment / Address Family Border Router. The goal is to make cost effective use of existing facilities and equipment where possible.

Current softwire solutions are mostly based on L2TPv2, which is defined in RFC 2661, "Layer Two Tunneling Protocol 'L2TP'", August 1999. L2TPv1 was defined in RFC 2341, "Cisco Layer Two Forwarding (Protocol) 'L2F'", May 1998. L2TPv2 is layered on PPP, which is defined in RFC 1661, "The Point-to-Point Protocol (PPP)", July 1994. All L2TPv2 connections use UDP encapsulation. There are already some very large deployments of softwires on LT2Pv2 in ISPs today. L2TPv2 meets all *IPv6 over IPv4* softwire requirements today. It is 99% ready for *IPv4 over IPv6* softwire today.

Future softwire solutions will be based on L2TPv3, which is defined in RFC 3931, "Layer Two Tunneling Protocol – Version 3 (L2TPv3)", March 2005. L2TPv3 *can* be layered on PPP, but in v3 it is optional (it can layer directly on IP). UDP encapsulation is also optional in v3. UDP encapsulation is useful for NAT traversal, but it increases overhead and lowers throughput and reliability. If no NAT needs to be traversed, turning off the UDP encapsulation can lower overhead. Session ID and Control Connection IDs are 32 bits (vs. 16 in L2TPv2). L2TPv3 also provides better user authentication, and data channel security through use of optional *cookies*. An L2TPv3 cookie is an up to 64 bit cryptographically generated random value, included in every packet. L2TPv3 is close to meeting all softwires requirements.

Here are the RFCs related to Softwires:

- **RFC 4935, "Softwire Problem Statement", July 2007 (Informational)**
- RFC 5512, "The BGP Encapsulation Subsequent Address Family Indicator (SAFI) and the BGP Tunnel Encapsulation Attribute", April 2009 (Standards Track)
- RFC 5543, "BGP Traffic Engineering Attribute", May 2009 (Standards Track)
- RFC 5549, "Advertising IPv4 Network Layer Reachability Information with an IPv6 Next Hop", May 2009 (Standards Track)
- **RFC 5565, "Softwire Mesh Network", June 2009 (Standards Track)**
- RFC 5566, "BGP IPsec Tunnel Encapsulation Attribute", June 2009 (Standards Track)
- **RFC 5571, "Softwire Hub and Spoke Deployment Framework with Layer Two Tunneling Protocol Version 2 (L2TPv2)", June 2009 (Standards Track)**
- RFC 5619, "Softwire Security Analysis and Requirements", August 2009 (Standards Track)
- RFC 5640, "Load-Balancing for Mesh Softwires", August 2009 (Standards Track)

There are a *lot* of IETF drafts related to softwires (this is a hot area of Internet development):

- draft-arifumi-softwire-dslite-global-addr-00, “Global IPv4 Address Configuration for DS-Lite”, 2010-03-01
- draft-boucadir-softwire-cgn-bypass-01, “Procedure to bypass DS-lite AFTR”, 2010-03-05
- **draft-cui-softwire-pet-02, “IPv4/IPv6 Coexistence Framework (PET)”, 2010-03-10**
- **draft-cui-softwire-pet64-00, “PET for IPv6 to IPv4 communication”, 2009-10-19**
- draft-despres-softwire-mesh-sam-01, “Stateless Address Mapping (SAM) – Mesh Softwires without e-BGP”, 2009-10-26
- draft-despres-sofwire-sam-00, “Stateless Address Mapping (SAM) for Softwire-Lite Solutions”, 2010-03-01
- draft-donnerhacke-softwire-IPv6-6to4-00, “More specific unicast routing for 6to4”, 2009-11-26
- **draft-gundavelli-softwire-gateway-init-ds-lite-03, “Gateway Initiated Dual-Stack Lite Deployment”, 2010-03-08**
- draft-gui-softwire-6rd-IPv6-config-00, “IPv6 Host Configuration in 6rd Deployment”, 2010-03-01
- draft-guo-softwire-auto-gre-00, “Auto GRE Tunnel for Hub and Spoke Softwire”, 2009-10-19
- draft-guo-softwire-sc-discovery-03, “Softwire Concentrator Discovery Using DHCP”, 2010-03-05
- draft-ietf-softwire-ds-lite-tunnel-option-02, “Dynamic Host Configuration Protocol for IPv6 (DHCPv6) Options for Dual-Stack Lite”, 2010-03-02
- **draft-ietf-softwire-dual-stack-lite-04, “Dual Stack Lite Broadband Deployments Following IPv4 Exhaustion”, 2010-03-08**
- **draft-ietf-softwire-IPv6-6rd-08, “IPv6 via IPv4 Service Provider Networks ‘6rd’”, 2010-03-23**
- draft-ietf-softwire-6rd-udp-00, “UDP Encapsulation of 6rd”, 2009-10-18
- draft-sarikaya-softwire-dslitemobility-01, “Dual-stack Lite Mobility Solutions”, 2009-10-20
- draft-wing-softwire-4over6multicast-02, “Port Control Protocol (PCP)”, 2010-03-08
- **draft-xu-softwire-4over6multicast-02, “Softwire Mesh Multicast”, 2010-03-08**
- draft-xu-softwire-tunnel-endpoint-01, “Simple Tunnel Endpoint Signaling in BGP”, 2010-02-22

PET (Prefixing, Encapsulation and Translation)

PET is one of the emerging softwire standards, which is trying to work out the optimal combination of tunneling and translation mechanisms to provide a workable framework for IPv4/IPv6 co-existence. The types of tunnels discussed are:

- IP-in-IP tunnels (RFC 2893, RFC 4213)
- GRE tunnel (RFC 1702)
- 6to4 tunnel (RFC 3056)
- 6over4 tunnel (RFC 2529)
- Softwire transition technique (RFC 5565)

The translation mechanisms discussed include:

- SIIT (RFC 2765)
- NAT-PT (RFC 2766 - deprecated)

- BIS (RFC 2767)
- SOCKS64 (RFC 3089)
- BIA (RFC 3338)
- IVI (draft-xli-behave-ivi)

The draft discusses various combinations of the above tunneling and translation mechanisms to accomplish different kinds of co-existence. The recommended tunneling scheme is Softwire transition technique (RFC 5565). It also notes that DNS may have to interact with the co-existence solution using a DNS Application Layer Gateway, such as DNS64.

7.5 – Translation

Translation between IPv4 and IPv6 is by far the most complex transition mechanism. It has all the issues of IPv4 to IPv4 Network Address Translation, plus new issues that complicate it even further. There is a great deal of activity in the IETF trying to create standards that will be implementable and deployable.

Since IPv4 addresses are running out, many ISPs would like to deploy IPv6-only service to their customers (as opposed to dual-stack with both IPv4 and IPv6 service). Without translation, an IPv6-only node cannot access legacy IPv4-only nodes on the First Internet (which currently includes most online sites). Over time, more and more sites and services will be dual-stack, which will make IPv6-only nodes more useful. Until that time, translation gateways will be needed for IPv6-only nodes. It will be far simpler and cheaper, resulting in a superior user experience if both IPv4 and IPv6 are deployed, even if the IPv4 service is heavily NATted. However, ISPs seemed to be obsessed with deploying translation. There are a variety of ways that this can be accomplished, but most are quite complex and likely to be major sources of problems.

Tunneling cannot achieve IPv4 to IPv6 interworking, but it's highly transparent and lightweight, can be implemented by hardware, and can keep IPv4 routing and IPv6 routing separated. It allows existing infrastructure (whether IPv4 or IPv6) to be used as a transport to link two nodes (or networks) using the other version of IP.

Translation achieves direct intercommunication between IPv4 and IPv6 nodes or networks by means of converting the semantics between IPv4 and IPv6. However it has limitations in operational complexity and scalability. Like any NAT, it may have serious issues with transparency (some protocols may not work through it). Correct translation requires:

- address or (address, port) tuple substitution
- MTU discovery
- fragmentation when necessary
- Translation of both IP and ICMP fields
- ICMP address substitution in payloads (e.g. with SIP)
- IP/TCP/UDP checksum recomputation
- Application Layer translation when necessary

Stateless translation consumes IPv4 addresses to satisfy IPv6 hosts, which does not scale (for one thing we are running out of IPv4 addresses, for another, there are lots more IPv6 addresses than IPv4 addresses). It can be implemented in hardware, but any ALG translation is too complex for hardware.

Stateful translation requires maintaining complex state for dynamic mapping of (address, port) tuples, and cannot be implemented in hardware.

Here are the RFCs related to IP layer translation:

- RFC 2765, "Stateless IP/ICMP Translation Algorithm (SIIT)", February 2000 (Standards Track)
- RFC 2766, "Network Address Translation – Protocol Translation (NAT-PT)", February 2000 (Standards Track – Deprecated)
- RFC 2767, "Dual Stack Hosts using the "Bump-In-the-Stack" Technique (BIS)", February 2000 (Informational)
- RFC 3089, "A SOCKS-based IPv6/IPv4 Gateway Mechanism", April 2001 (Informational)
- RFC 4787, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP", January 2007 (Best Current Practices)
- RFC 4864, "Local Network Protection for IPv6", May 2007 (Informational)
- RFC 5382, "NAT Behavioral Requirements for TCP", October 2008 (Best Current Practice)
- RFC 5508, "NAT Behavioral Requirements for ICMP", April 2009 (Best Current Practice)

Again, there are many Internet Drafts related to IP layer translation:

- draft-ietf-behave-v6v4-xlate-18, "IP/ICMP Translation Algorithm"
- draft-xli-behave-ivi-07, "The CERNET IVI Translation Design and Deployment for the IPv4/IPv6 Coexistence and Translation", 2010-01-06
- draft-ietf-behave-dns64-09, "DNS64: DNS extensions for Network Address Translation from IPv6 Clients to IPv4 Serves", 2010-03-30
- draft-ietf-behave-v6v4-xlate-stateful-11, "Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers", 2010-03-30
- draft-ietf-behave-address-format-07.txt, "IPv6 Addressing of IPv4/IPv6 Translators", 2010-04-09
- draft-ietf-behave-v6v4-framework-08, "Framework for IPv4/IPv6 Translation", 2010-03-22
- draft-haddad-mext-nat64-mobility-harmful-00, "A Note on NAT64 Interaction with Mobile IPv6", 2009-10-11
- draft-chen-behave-olnat-01, "A Optimal Load-balance mechanism for NAT64 (OL-NAT)", 2009-10-26
- draft-wang-behave-nat64-load-balancer-01, "Some Considerations on the Load-Balancer for NAT64", 2010-02-21
- draft-wing-behave-nat64-referrals-01, "Referrals Across an IPv6/IPv4 Translator"

RFC 4864, "Local Network Protection for IPv6" makes a very strong case against use of NAT in IPv6, which was designed with the express intention of making NAT unnecessary. It shows how simple IPv6 mechanisms can provide all of the perceived benefits of NAT in existing IPv4 networks, without recourse to NAT.

RFC 2765, “Stateless IP/ICMP Translation Algorithm (SIIT)”, is used in NAT-PT, IVI and NAT64/DNS64. It is not a candidate itself for v6->v4 translation, but is included to help understand other more complete mechanisms.

RFC 2766, “Network Address Translation – Protocol Translation (NAT-PT)”, is not a viable candidate, as so many problems have been encountered that it has been officially deprecated to historic status by the IETF.

RFC 3089, “A SOCKS-based IPv6/IPv4 Gateway Mechanism” is not a viable candidate, because it requires “socksification” (modifications to the source code to insert a SOCKS layer between all system calls and the socket API) of all software applications that need to use the gateway.

IVI (currently described in draft-xli-behave-07) is a potentially viable candidate for v6->v4 translation, as it has been used extensively in CERNET in China, to allow IPv6-only clients to access IPv4-only servers. Since this mechanism is still in Internet Draft form, it should be considered experimental, and not for deployment in critical infrastructure.

NAT64/DNS64 (currently described in draft-ietf-behave-v6v4-xlate-stateful-11 and draft-ietf-behave-dns64-09) is a potential candidate for v6->v4 translation, and an open source implementation is available from gogo6. Since these mechanisms are still in Internet Draft form, they should be considered experimental, and not suitable for deployment in critical infrastructure.

7.5.1 – NAT64 / DNS64

This translation mechanism requires both a NAT64 gateway and either a DNS server that supports DNS64 mapping, or a DNS ALG that supports DNS64. What follows is a highly simplified description of operation. The full details are covered in the Internet Draft (there is quite a bit of complexity involved in the real operation).

The NAT64 gateway should have two interfaces, one connected to the IPv4 network (with a valid IPv4 address on that network), and the other connected to the IPv6 network (with a valid IPv6 address on that network). IPv6 traffic from a node on the IPv6 network going to an IPv4 node is sent in IPv6 and routed to the NAT64 gateway. The gateway does address translation and forwards the translated packets to the IPv4 interface, from which it is routed to the destination node. Reply packets from the IPv4 node are sent in IPv4 to the gateway, are translated into IPv6 and forwarded to the IPv6 interface, from which they are routed back to the original sender. This process requires state, binding an IPv6 address and TCP/UDP port (referred to as an IPv6 transport address) to an IPv4 address and TCP/UDP port (referred to as an IPv4 transport address).

Packets that originate on the IPv4 side cannot be correctly translated, because there would be no state from the packets coming through the gateway in the v6->v4 direction. NAT64 is not symmetric. For traffic initiated by an IPv6 node, everything works right. Once the binding is created, that traffic flow can continue (from the IPv6 node to the IPv4 and back).

For the traffic originating on the IPv4 side to be translated to IPv6 requires some additional mechanism, such as ICE or a static binding configuration.

This mechanism depends on constructing *IPv4-Converted IPv6 addresses* (defined in draft-ietf-behave-address-format-nn). Each IPv4 address is mapped into a different IPv6 address by concatenating a special IPv6 prefix assigned to the NAT64 device (Pref64::/n).

It also uses a small pool of IPv4 addresses, from which mappings will be created and released dynamically, as needed (as opposed to permanently binding specific IPv4 addresses to specific IPv6 addresses). This implies that NAT64 does both address and port translation.

When an IPv6 initiator does a DNS lookup to learn the address of the responder, DNS64 is used to synthesize AAAA resource records from A resource records. The synthesized AAAA resource records are passed back to the IPv6 initiator, which then initiates an IPv6 connection with the IPv6 address that is associated with the IPv4 receiver. The packet will be routed to the NAT64 device, which will create the IPv6 to IPv4 address mapping as described before.

In general dual-stack nodes should not use DNS64. If they get a synthesized IPv6 address *and* a native IPv4 address, the rule to prefer IPv6 will cause the dual-stack host to do the access via the NAT64 gateway instead of direct using IPv4. If you deploy DNS64, it should be used *only* by IPv6-only nodes, and there should be a regular DNS for use by any dual-stack nodes.

7.5.2 – IVI

This address translation scheme is being used on a large scale between CERNET (IPv4-only) and CERNET2 (IPv6-only) to nodes on either side to connect to nodes on the other side, as well as allowing IPv6-only nodes connect to IPv4 nodes out on the public Internet.

The pros of using IVI are as follows:

- It is stateless, so it scales to large number of nodes better than NAT64 / DNS64.
- The translation is decoupled from DNS.
- It is symmetric, so can be used for connections initiated on either side of the gateway (IPv4 to IPv6 side, or IPv6 to IPv4 side).
- There is an open source implementation of the IVI gateway and DNS64 ALG available on Linux.

The cons of using IVI are as follows:

- An ALG is still required for any protocol that embeds IP addresses in the protocol, such as SIP.
- It restricts the IPv6 hosts to use a subset of the addresses inside the ISP's IPv6 block. Therefore, IPv6 Stateless Address Autoconfiguration cannot be used to assign IPv6 addresses to nodes. You must either manually assign addresses, or use stateful DHCPv6.
- There are still some issues with end-to-end transparency, address referrals and incompatible semantics between protocol versions.
- You still need a DNS64 ALG for DNS.

7.6 – Recommendations on Transition Mechanisms

There are so many problems with scalability, stability, transparency, and need for nonstandard infrastructure (e.g. DNS64), possibly not being able to use Stateless Address Autoconfiguration, that if at all possible, you should avoid using translation. It is not difficult to deploy dual-stack in your LAN (either using conventional Dual Stack, or Dual-Stack Lite if that is what your ISP provides). The complexity from running two parallel networks (IPv4 and IPv6) on the same infrastructure is minor compared to the issues from using translation. If direct dual stack ISP service is available, then definitely you should deploy dual-stack in your network. If not, you can easily tunnel in IPv6 service to a gateway router/firewall and route it into your network alongside existing IPv4.

If you restrict those protocols that don't work well with NAT to run only on the IPv6 side (e.g. SIP, IPsec VPN, P2P, etc.), then the IPv4 side can have addresses that are heavily (even multiply) NATted, so there is no problem finding addresses for the IPv4 side, even after the IPv4 address space is depleted. Legacy applications that work well even over NAT can be run over the IPv4 side until there are time and resources available to migrate them to IPv6.

Virtually all modern operating systems support dual-stack today. Most Microsoft products (client and server) have good support for dual-stack. Most open-source applications support it very well.

All standard infrastructure can be used – no need for NAT specific variants of things like DNS.

If you *have* to use translation, IVI is probably the least of several evils, and has a track record of working well on a large scale in CERNET / CERNET2 in China. You should plan on using stateful DHCPv6 to assign addresses (since Stateless Address Autoconfiguration won't work). You should deploy client nodes with *only* IPv6, and depend on access via the IVI gateway for any IPv4 access. You will still have issues with SIP over IPv4, so you should either plan on using SIP only over IPv6, or deploy a SIP ALG to resolve the issues with addresses in the protocol.

Chapter 8 – DNS

DNS (the Domain Name System) is a critical part of today's Internet. Without it, we would have to keep massive (and always out-of-date) directories (like telephone books), where you could look up the name of some site (such as Dell's pages about their PCs), then find the "telephone number" (IP address) of that page, which you would then "dial" (type into your browser). This is clearly not very practical. DNS is such a complex and critical topic for both TCP/IPv4 and TCP/IPv6 that I have included a chapter just for it.

8.1 – How DNS Evolved

Various schemes have been used to keep track of node names and their corresponding IP addresses. The end result is a remarkably powerful and flexible system called DNS.

8.1.1 – Host files

In the early days of TCP/IP, a list of known "hosts" (nodes) was kept in a special file called *hosts* in the */etc* directory of all UNIX computers (the complete filename was */etc/hosts*). This file included one or more lines, each of which contained an IP address, followed by one or more nodenames (e.g. www) or even fully qualified nodenames (e.g. www.ibm.com). If you told your copy of UNIX to use the hosts file for "name resolution", when you used one of the nodenames listed in your *hosts* file, it would use the IP address associated with that name. This still works today – you can override DNS with your *hosts* file if you specify it first in the search order. Even Windows systems have a *hosts* file, typically located at

C:\Windows\System32\drivers\etc\hosts

A typical *hosts* file might look like:

```
172.20.0.11 ws1 ws1.hughesnet.local
172.20.0.12 ws2 ws2.hughesnet.local
172.20.0.13 us1 us1.hughesnet.local
```

8.1.2 – Network Information Service (NIS)

In organizations with many UNIX computers, and especially once people started linking networks together with TCP/IP, it became necessary to keep everyone's *hosts* file up to date and synchronized. This was done manually for a while, then NIS (Network Information Service) was created by Sun to automatically distribute copies of the official *hosts* file (in addition to other important configuration files for UNIX) to every node, on a periodic basis.

8.1.3 – DNS is invented

Soon even this became unwieldy, so in 1983, at the request of Jon Postel, Paul Mocapetris designed DNS as a distributed database engine with distributed data. We are still using this system today. You will find that with minor extensions, it even supports IPv6 and Dual Stack networks. There is a gigantic, worldwide hierarchical system of DNS servers that allows each network administrator to manage the names and IP addresses of nodes in their network that users anywhere in the world might need to know about (e.g. that organization's web servers, E-mail servers, etc.). DNS is also used to keep track of the nodenames and IP addresses of all *internal* nodes in an organization's private network, which only users in that organization need to know about (the organization's file servers, network printers, intranet web servers, other user's workstations, etc.).

8.2 – Domain Names

Domain names refer to the hierarchical name space as defined by RFC 1035 (above); RFC 1123 "Requirements for Internet Hosts – Application and Support", October 1989; and RFC 2181, "Clarifications to the DNS Specification", July 1997. Briefly, domain names consist of a list of names (e.g. *atlanta*, *usa*, *exampleco* and *com*, in most specific to least specific order, separated by periods (e.g. *atlanta.usa.exampleco.com*). Here, *com* is the TLD (or Top Level Domain) name for commercial organizations. The name *exampleco* is the name of a hypothetical company, which is a commercial organization, and all parts of it use the domain *exampleco.com*. Within ExampleCo, there is a branch in the U.S., which uses the subdomain *usa.exampleco.com*. Finally, there is an office in Atlanta GA, which uses the subdomain *atlanta.usa.exampleco.com*. If there is a web server named *www* in that office, it would have a fully qualified domain name of *www.atlanta.usa.exampleco.com*. There is no way to tell (without more information) if the first name in such a string is a node's name or the first component of a domain name.

8.2.1 – Top Level Domain Names

There are a number of TLDs including *generic* ones that have been in use for a long time:

com	for commercial organization (company)
org	non-commercial organization
edu	Educational organization
net	Internet related, e.g. ISP
gov	Government related
mil	Military related

There are also many ccTLDs (country code Top Level Domains), each of which uses the ITU two letter code for the country, such as *us*, *uk*, *jp*, and *ph*). There are a few exceptions to the ITU code usage, e.g. the ITU code for Great Britain is *gb*, while their ccTLD is *uk*. Each country manages subdomains under their ccTLD as they best see fit. Certain ccTLDs appear to have other meanings, like *tv* for the country of Tuvalu, which sells domains in their space to people who want to use it to mean *television*. Under ccTLDs, there are usually (but not always) second level domains, such as *co* for commercial, *or* for organization, etc. Actual organization names would then be *third* level domain names. Hence a UK based

commercial entity called Warmbeer, Ltd. might have the domain name *warmbeer.co.uk*. Their web server might be *www.warmbeer.co.uk*. A few ccTLDs, like *ph* for the Philippines use the full three letter code for organization type instead of the more common two letter codes, as second level domain name. For example, our Philippine domain name is *infoweapons.com.ph*.

Recently, some new generic TLDs have been introduced, including *info*, *ws*, *museum*, *aero*, *asia*, *biz*, *coop*, *int*, *mobi*, *name*, *pro*, *tel* and *travel*. None of those have become really popular yet. The main reason people use them is that they can't get the domain name they want using the more common TLDs like com, org or net.

8.2.2 – Internationalized Domain Names

There are also *internationalized domain names* (IDNs) that use 16-bit Unicode characters to allow domain names in languages that have non-Latin alphabets. Your browser will translate these Unicode domain names into strings in UTF8, using the *punycode* algorithm (shown in the last column below). This is defined in RFC 3492, "Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications (IDNA)", March 2003. For example, the following (believe it or not) are syntactically valid URLs (although they do not current point to real sites):

http://пример.испытание/	Russian	http://xn--e1afmkfd.xn--80akhbyknj4f/
http://例子.测试/	Chinese	http://xn--fsqu00a.xn--0zwm56d/
http://실례.테스트/	Korean	http://xn--9n2bp8q.xn--9t4b11yi5a/
http://例え.テスト/	Japanese	http://xn--r8jz45g.xn--zckzah/
http://□□□□□□.□□□□□□□□/	Indian	http://xn--p1b6ci4b4b3a.xn--11b5bs3a9aj6g/

8.3 – DNS Resolver

All operating systems today include a DNS client, called a *resolver*. All network applications use the resolver to look up nodenames and obtain their corresponding IP addresses, whether those nodes are local (on the organization LAN) or external (out on the Internet). The resolver contacts one of the DNS servers specified in their TCP/IP configuration (either local, or at your ISP). If that server is *authoritative* for the requested domain names, it returns the addresses immediately. Otherwise, that server can either return a hint of where to look ("I don't have that information.. try here") or do a recursive lookup ("I didn't have that information, but I went and found it for you"). Of course, the lookup could fail ("I couldn't find that domain name anywhere").

8.4 – DNS Server Configuration

The full process of setting up DNS servers (usually two or more) for an organization and populating them with node information is too complicated to cover in this book. If you are using the Microsoft DNS server (included free with Windows Server), see their documentation for details. If you are using BIND (the

freeware DNS server from the Internet Software Consortium), see O'Reilly's "BIND and DNS, 5th Edition" for details. If you have a DNS appliance, consult their documentation or online help for details.

In general though, you define both "forward zones" that map nodenames to IP address, and "reverse zones" that map IP addresses onto nodenames. You also have to inform all client computers of the IP addresses of at least two DNS servers, that they can use for resolving nodenames to IP addresses (or vice versa). Client computers can be informed of these DNS server addresses either via manual configuration, or automatically via DHCPv4 or DHCPv6. If your client computer doesn't know where to find DNS servers, you may have full Internet connectivity but no name resolution. You can ping (or even surf to) nodes anywhere in the world by specifying their numeric IP addresses (e.g. *http://64.170.98.32* – try it!). However, most people would consider such a computer to not be very useful. This gives you a very good idea of how important DNS is to the Internet (even the Second Internet).

8.5 – DNS Protocol

DNS is an *Application Layer* protocol. It uses UDP port 53 (for most queries and responses) or TCP port 53 (for zone transfers between DNS servers). It was originally defined in RFC 882, "Domain Names – Concepts and Facilities", November 1983, and RFC 883, "Domain Names – Implementation and Specification", November 1983. Those were replaced by RFC 1034 "Domain Names – Concepts and Facilities", November 1987, and RFC 1035, "Domain Names – Implementation and Specification", November 1987. There have been numerous updates to these, including RFCs 1101, 1183, 1348, 1876, 1982, 1995, 1996, 2065, 2136, 2137, 2181, 2308, 2535, 2845, 3425, 3658, 4033, 4034, 4035, 4343 and 4592.

8.6 – DNS Resource Records

The data in DNS servers is kept in *resource records*. In forward zones, it is possible to have any of the following resource records (the following list is not comprehensive):

Name	Contents
A	"A" - IPv4 address associated with a domain name
AAAA	"Quad-A" - IPv6 address associated with a domain name
MX	"Mail eXchange" – domain name of a mail server for the domain
SRV	"Service" – domain name of servers for other protocols, such as SIP and LDAP
CNAME	"Alias" – provide an alternative domain name for another domain name
HINFO	"Host Info" – any arbitrary info you want to provide about a host
NAPTR	"Naming Authority Pointer" – used mostly in ENUM
NS	"Name Server" – name of a valid DNS server for this domain
SOA	"Start of Authority" – start of a zone in configuration files, includes default TTL
SPF	"Sender Policy Framework" – used in anti-spam technology
TSIG	"Transaction Signature" – symmetric cryptographic key used in zone transfers
TXT	"Text" – Any arbitrary text information (not interpreted by DNS)

In reverse zones, typically only the following resource records are found:

NS	"Name Server" – name of a valid DNS server for this domain
SOA	"Start of Authority" – Same as in forward zones
PTR	"Pointer Record" – IP address for a specific node, in reverse order

The following examples show how typical resource records look:

```
ws1             IN  A       172.20.0.11
ws1             IN  AAAA    2001:418:5403:3000::c
                IN  MX      10 ws1.hughesnet.org
11.0.20.172     IN  PTR     ws1.hughesnet.org
c.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.3.3.0.4.5.8.1.4.0.1.0.0.2    IN  PTR     ws1.hughesnet.org
```

In general, it is a pain to manually create reverse PTR records, and any change to IP addresses (e.g. from changing ISPs) requires changes to all forward and reverse resource records in DNS. Here again, an appliance with GUI can help by automatically generating reverse PTR resource records. This is especially useful for IPv6 reverse PTR records.

In the InfoWeapons SolidDNS appliance, you can define *named networks* for IPv6. When you define a network, which will create the associated reverse zone, you can assign that network a name, which has the value of the network's prefix. You can then define node addresses in terms of the network name. First, this fills in the first 64 bits of each address, which reduces errors and saves time. However, if you ever change ISPs, you can simply redefine the prefix for the network, and all forward and reverse resource records created from the nodes specified using that network name will be updated with the new prefix. This is called *instant prefix renumbering*. There was an *a6* resource record created at one point for IPv6 forward resource records that was supposed to accomplish this, but there were so many problems with it, it has now been deprecated (you are not supposed to use it any more). It is much better to do this in an appliance that has GUI and database, and generates only the standard AAAA resource records.

8.7 – DNS Servers and Zones

A given DNS server can have any number of *zones* defined on it. A given zone can be a *forward zone* (for mapping domain names to IP addresses) or a *reverse zone* (for mapping IP addresses to domain names). There is usually one forward zone for each *domain* for which the DNS server contains information (e.g. *hughesnet.org*), and one reverse zone for each *network* that DNS server contains information for (e.g. *172.20.0.0/16*). So, the forward zone for hughesnet.org might contain mappings for *ws1.hughesnet.org* to *172.20.0.11*, *for us1.hughesnet.org* to *172.20.0.13*, and so on. The reverse zone for *172.20.0.0/16* might contain mappings from *172.20.0.11* to *ws1.hughesnet.org*, from *172.20.0.13* to *us1.hughesnet.org*, *172.20.0.91* to *us1.v6home.org*, and so on.

Any zone (forward or reverse) can be a *primary zone* or a *secondary zone*. A primary zone is one on which the DNS administrator manages the contents of (either via a GUI interface or via editing BIND configuration files). A secondary zone is one whose contents are automatically transferred from a corresponding primary zone of the same name on a different DNS server (no management is required for a secondary zone, once that zone is created). When you create a secondary zone, you specify the IP address of the DNS server that contains the corresponding primary zone. Usually there is one primary

zone (on one DNS server) and one or more secondary zones (each on other DNS servers) for a given set of records. A given DNS server can have any mix of primary zones and secondary zones. Sometimes the terms *primary* and *secondary* are used for entire DNS servers, especially if all zones on a server are all primary zones or all secondary zones, but technically the terms refer to zones, not servers. The transfer of all records from a primary zone on one DNS server to a secondary zone on another DNS server is called a *zone transfer*. Typically a primary zone is configured to allow zone transfers only to secondary zones on authorized DNS servers (by IP address). There is also a cryptographic authentication scheme called TSIG that can restrict zone transfers to only authorized secondary zones. Otherwise, a hacker could perform a zone transfer from one or more of your primary zones, and obtain information useful in attacking your network (effectively, a "map" of at least part of your network). Typically zone transfers from primary zones to secondary zones are done automatically on a periodic basis. If a hacker changes data in a *secondary* zone, the correct data would be automatically restored as of the next zone transfer. If a hacker changes data in a *primary* zone, the hacker's changes will be automatically and securely transferred to all secondary zones via the regular zone transfers. It is *very* important to secure your primary zones.

It is possible for all of the zones on a given DNS server to be accessible by one or more clients for performing DNS resolutions (lookups), in which case it is a *resolving* server. A primary server that is not accessible for resolutions by any client (or other DNS servers) is called a *stealth server*. It is only ever used to do zone transfers to secondary servers (hence need not be very powerful). Access via UDP port 53 can be completely disabled (zone transfers take place over TCP port 53), and even those can be restricted by IP address. Use of a stealth server lowers the possibility of hackers being able to attack your primary DNS server. There would be no real use for a "stealth secondary server".

8.8 – Different Types of DNS Servers

There are different types of DNS servers based on how they are populated with data.

8.8.1 – Authoritative DNS Servers

A DNS server which contains a primary zone or a secondary zone is said to be *authoritative* for the domain (or network) defined in that zone. All resolving servers cache (temporarily store) the results of any query they perform on behalf of clients.

If a client makes a query of a resolving server that currently has the required information (either because it is authoritative or because it has cached it from a previous query) it responds with that information to the client immediately. If a resolving server is asked for information it does not currently have, it can either return a *reference* ("I don't know, go ask *this* server"), or it can do a *recursive* query on the client's behalf ("I didn't know, but I went and found out for you by making client queries myself on your behalf, and here is what I found.") A recursive query can go through several servers before the requested information is finally obtained and returned to the client that asked for it in the first place. Any server involved in the process typically caches the retrieved information. Every record published by a DNS server has a *Time To Live* (TTL) defined for it. When a record is cached, it is kept on the caching server only for the defined *Time To Live* for that record, after which it is considered *stale* and is discarded. Once a DNS server discards stale information, if it is asked for it again, it must do another

recursive query, at which point it again caches the record. This caching and expiration scheme keeps the data current, but means that a change to authoritative information may take a while to *propagate* to all other servers (often 24 to 48 hours, depending on *Time To Live* values chosen).

When a client obtains information from an authoritative server, it is reported as an authoritative answer. When it obtains information that has been cached, it is reported as a non-authoritative answer. This doesn't mean it is any less trustworthy, just that it obtained the information at "second hand", (out of some DNS servers cache) instead of directly from the authority on the subject (an authoritative server).

8.8.2 – Caching-Only Servers

A resolving server that has no defined primary or secondary zones, is called a *caching only* server, and typically, once setup and configured, requires little or no management.

8.9 – Client Access to DNS

In a typical network, every client should have the addresses of at least two valid resolving DNS servers configured. If a connection to one of them fails, the client will automatically try the other configured addresses. This increases the robustness of the network. In a small network (e.g. home connection), the specified servers may be located at and managed by the ISP. In some cases, the DSL or cable modem might provide a *DNS proxy* function, which allows DNS queries to be submitted to the default gateway address. The modem relays such requests to the DNS servers configured in the modem, and returns the replies to the internal client that made the request.

Any network can have one or more *local* DNS servers (assuming they can make outgoing queries via UDP port 53). To run an authoritative server on a network, that server must be accessible by relevant clients and other servers. If any of those clients or servers is *external*, then the authoritative server must have a globally accessible "external" IP address (not a *private* IP address). For example, I run an authoritative DNS server for my domain *hughesnet.org* in my home, on a DNS server that has a valid external IP address. I also run other servers (E-mail and web) that have globally accessible *external* IP addresses (in my case, *both* IPv4 and IPv6 addresses). I can access these services from anywhere on the Internet, just like using servers at ISPs. This sort of thing is far simpler and less expensive with IPv6 than with IPv4.

8.9 1 – Recursive DNS Queries

A single DNS query (e.g. "lookup the IP addresses for node *ws1.hughesnet.org*") can actually require several resolutions. If the server already has information for *ws1.hughesnet.org* , either because it is authoritative for that information, or because it has still valid cached information, it returns the requested information immediately ("*ws1.hughesnet.org* has an IPv4 address which is *172.20.0.11* and an IPv6 address which is *2001:df8:5403:3000::c*"). It is up to the client which of these is used. If it is a dual-stack client (supports both IPv4 and IPv6) it should use the IPv6 address by preference.

If information for *ws1.hughesnet.org* is not present on the resolving DNS server, that server must find the authoritative server for the domain *hughesnet.org*. The server that is authoritative for the domain *org* can tell it this information. To locate *that* server, the resolving server can ask any *root* DNS server. To locate a root DNS server, the resolving server can look in its *root hints* file. Any of these things could already be in cache (and typically are, if any other nodename ending in .org or .hughesnet.org have been looked up by any client recently). If none of them are in cache, then first, the resolving DNS server will ask a root DNS server "who is authoritative for domain *org*". It will cache the response it gets, and ask the returned server for *org* "who is authoritative for domain *hughesnet.org*". It will cache that response also, and ask the returned server for *hughesnet.org* "what is the IP address of *ws1.hughesnet.org*". It will cache that response as well, and return the answer to the client, who has been patiently waiting. Most DNS servers have a way to empty (or "dump") the cache if you would like to watch all of this happen with a network sniffer (this would require root level access on the computer running your DNS server).

8.10 – The Root DNS Servers

All DNS queries eventually chain up to one of the 13 *root DNS servers* (or the cached data from them). In reality, "DNS anycast" is employed so that there are actually quite a few copies of most of the 13 root servers distributed around the world (see table below). The current information on the root servers (from which I made the table below) is always available at:

http://www.root-servers.org

Every DNS server includes a file with the current anycast addresses of the 13 root servers (a.root-servers.net to m.root-servers.net), as summarized in the table below. A copy of the official current file (in BIND format) can always be found at:

http://www.internic.net/zones/named.root

All DNS server operators from time to time obtain the current copy of this file and update their server(s) *root hints* file with it. The information in this file allows a DNS server to locate a root server when it needs one.

The only thing the DNS root servers publish is the information in a short file that is maintained by IANA that helps other DNS servers locate the DNS hierarchy layer just below that of the DNS root servers (i.e. the servers that are authoritative for the top level domains such as *com*, *net*, *org*, *uk*, *jp*, etc.) All root servers publish the same information, so only one ever needs to be asked (typically chosen at random from the 13 available). A copy of the current version of this information (in BIND format) can always be found at:

http://www.isoc.org/briefings/020/zonefile.shtml

Only operators of DNS root servers ever actually need to obtain this file and update their DNS root servers with it. In reality, due to DNS caching the actual root servers are only rarely involved in a typical DNS query. A typical non-root DNS server only needs to access a root server about once every 48 hours. It would normally have the information published by the root servers cached in memory from previous

enquires. Only once the *Time To Live* expires for a given resource record obtained from an actual root server, would the DNS server have to go back and obtain more up to date information from a DNS root server (which it would again cache to use in future lookups). Most of the time, this new information will just be the same information that just expired.

Current Root Servers (all in the domain "root-servers.net")

Name	Organization	Count	IPv4 Address	IPv6 Address
A	VeriSign, Inc.	6/3	198.41.0.4	2001:503:ba3e::2:30
B	Information Sciences Inst.	1/1	192.288.79.201	2001:478:65::53
C	Cogent Communications	6/0	192.33.4.12	-
D	University of Maryland	1/0	128.8.10.90	-
E	NASA Ames Research Center	1/0	192.203.230.10	-
F	Internet Systems Consort.	49/22	192.5.5.241	2001:500:2f::f
G	U.S. DoD Network Info Ctr	6/0	192.112.36.4	-
H	U.S. Army Research Lab	1/1	128.63.2.53	2001:500:1::803f:235
I	Autonomica	34/0	192.36.148.17	-
J	VeriSign, Inc.	70/6	192.58.128.30	2001:503:c27::2:30
K	RIPE NCC	18/10	193.0.14.129	2001:7fd::1
L	ICANN	3/3	199.7.83.42	2001:500:3::42
M	WIDE Project	6/5	202.12.27.33	2001:dc3::35

In the above table, the first number in the count field (before the slash) is the total number of anycast servers for that name, regardless of IP version. The second number (after the slash) is the number of anycast servers for that name that can accept queries over IPv6. Currently, all root servers will accept queries over IPv4 (this may not always be the case). All root servers can return A and/or AAAA records for the servers authoritative for Top Level domains. One of the watershed events for IPv6 happened in February 2008, when VeriSign enabled IPv6 access on enough of the root servers that a client doing queries over IPv6 would always be able to complete a query without having to fail back to IPv4. Since then, clients that access DNS over IPv6 (IPv6-only nodes or dual stack nodes) can resolve names to addresses as effectively as IPv4-only nodes have been able to since the introduction of DNS. Eventually all root servers will support queries over IPv6.

Total Root Server Names = 13
Total Root Server Names that accept connections over IPv6 = 8 (61.5% of names)
Total Deployed Root Servers = 202
Total Deployed Root Servers that accept connections over IPv6 = 51 (25.2% of total)

8.11 – MX and SRV records

In addition to providing nodename to IP address lookup (forward resolution) and IP address to nodename lookup (reverse resolution), DNS servers can also advertise the preferred servers for various functions, such as E-mail (SMTP), VoIP (SIP), etc.

The MX (Mail eXchange) record can advertise one or more E-mail server names, with priorities. Other mail servers when they want to deliver mail to your domain will do a DNS query asking for the MX record(s) for your domain. The sending server will try to make connections over port 25 (SMTP) to the advertised nodenames, in decreasing priority, until it either has a connection accepted (in which case it

will deliver all the mail it has for your domain), or it runs out of advertised nodenames (in which case it will try again on some schedule, until it succeeds, or decides your domain is not currently online). Thus a client can send messages to any name at your domain (fred@hughesnet.org), and it will be delivered to one of your preferred mail servers, which will then deliver it to that person's mailbox, or return it as undeliverable.

The SRV record (defined in RFC 2782, "A DNS RR for specifying the location of services (DNS SRV)", February 2000) can be used to specify preferred servers for your domain for services other than E-mail, such as VoIP (SIP), Jabber Instant Messaging (XMPP), and Directory Services (LDAP). In the same style as E-mail addresses, it is possible to specify a new style "phone number" as a SIP URI, e.g. *sip:fred@hughesnet.org*. A good SIP client could take that URI, do a DNS SRV query to determine the preferred SIP server for the domain *hughesnet.org*, then attempt to connect to user *fred* on that server. The SRV mechanism of DNS will make VoIP as scalable and decentralized as the current Internet E-mail system is. The same mechanism would allow clients anywhere in the world to contact your Jabber IM client (*im:fred@hughesnet.org*), or retrieve your directory entry from your organization's external LDAP server.

8.12 – ENUM

During the transition from legacy telephony service (using ITU E.164 numeric phone numbers) to more general SIP URIs, there is a need for a transition mechanism. One has been developed called ENUM (which stands for *E.164 Numbers*, where E.164 is the ITU standard for conventional numeric telephone numbers), including the international country codes (e.g. +1 for U.S., +63 for Philippines, +852 for Hong Kong). ENUM is implemented as a sub function of DNS.

With ENUM, you do the equivalent of reverse DNS lookups, but from ITU telephone numbers to one or more URIs, typically including a SIP URI. This allows people with legacy telephone customer equipment that only has a 10 key pad to map E.164 telephone numbers onto the complex alphanumeric SIP URIs (e.g. *sip:fred@hughesnet.org*). In addition to SIP URIs, you can also map a single E.164 number onto other URIs, including instant messaging (*im:*), email (*mailto:*), web (*http:*), etc. This would allow a smart client that supports unified messaging via various protocols to contact you via any of your addresses using a single numeric "phone number". These URIs are associated with an E.164 number using the NAPTR DNS resource record. In theory, the long list of various kinds of "addresses" on your business card whereby people can contact you (telephone, fax, E-mail, im, web, etc.) could be replaced with a single ENUMber. There are standalone ENUM clients that can allow someone to simply view the mappings for any ENUMber, for use with devices such as legacy telephone handsets.

Essentially, you dial a person's ENUMber, then your client (or your VoIP server) does a DNS query to map that onto one or more URIs, which it then uses to contact that person. ENUM can be run at the national level, typically by a large telco or a government agency, using ENUMbers that start with the actual country code. In this case, the general public DNS system is used to do the mapping. It can also be done on a private basis, using any "country code" you want (even made up ones), but requires clients to specify a custom DNS server to do the custom mapping (all ENUM compliant clients allow specification of which DNS server to use for ENUM lookups).

If traditional DNS management is used to manage the URI mappings for a large number of people, this could be an enormous workload and lead to long delays in making additions or changes to URI mappings. It is better to allow the end users to view and modify their own mappings. This requires user authentication, which most often would be linked to an existing authentication server, such as Microsoft Active Directory (within an organization) or RADIUS (for a telco or ISP).

Because most phones from 3.5G onward will be based on IPv6, it is important that an ENUM capable DNS server support queries over IPv6. There will be some legacy phones that still use IPv4 for some time, so it must also support queries over IPv4. In other words, the DNS server used for ENUM must be dual-stack.

ENUM is defined in RFC 3761, "The E.164 to Uniform Resource Identifiers (URI) Dynamic Delegation Discovery System (DDDS) Application (ENUM)", April 2004.

There is more information on ENUM on the following websites:

http://www.enum-forum.org
http://www.ripe.net/enum
http://www.itu.int/infocom/enum/

8.12 – DNSSEC (Secure DNS)

As mentioned before, it is possible for a hacker to tamper with the information in the DNS system. You may think you are surfing to your bank's website, but it could be a clever mockup in some hacker's basement. It will show the correct URI in your browser, but anything you input (like your login) can be easily captured by the hacker, and then used to gain access to your account on the real site. A hacker could also trick people into reading fake news stories that appear to be from legitimate websites.

There are some patches to BIND that make it more difficult for a hacker to alter DNS data, but the only surefire way to detect it is to add digital signatures into the DNS system. The following RFCs define such an extension to DNS, which is called DNSSEC.

- RFC 3833, "Threat Analysis of the Domain Name System (DNS)", August 2004
- RFC 4933, "DNS Security Introduction and Requirements", March 2005
- RFC 4034, "Resource Records for the DNS Security Extensions", March 2005
- RFC 4035, "Protocol Modifications for the DNS Security Extensions", March 2005
- RFC 4431, "The DNSSEC Lookaside Validation (DLV) DNS Resource Record", February 2006
- RFC 4641, "DNSSEC Operational Practices", September 2006
- RFC 5074, "DNSSEC Lookaside Validation (DLV)", November 2007

Currently, various parts of the DNS namespace (the *org* domain, the *gov* domain, various ccTLDs) are being secured with DNSSEC. Once a Top Level Domain is signed, this simplifies signing of domains under it. Eventually the root of the entire DNS tree (".") will be signed, and this will unify DNSSEC for all domains. Until that time, there will be a need for DNS servers to obtain the root key material for any part of the DNS space for which you want to verify signatures.

To review cryptography and PKI, see appendix A. DNSSEC depends heavily on digital signatures and public key digital certificates. You will need to understand these concepts to follow this discussion. Basically though, a digital signature is created by a cryptographic algorithm that produces a numeric result that is derived from a specific plaintext (in this case, a single DNS resource record) and an asymmetric cryptography *private key*. These signatures are generated, encoded into ASCII characters, and inserted into the BIND configuration files after each resource record. When DNS data are retrieved, the digital signatures flow along with the resource records. The user's resolving DNS server can verify the signature of any retrieved records by using the *public key* associated with the *private key* used to sign the records. Only if the signature verifies does the resolving server return the retrieved record(s) to the client. If records are compromised, they appear to not be available to the user. Eventually all client software will contain code to verify the DNSSEC signatures, in which case the resolving DNS server will just return the records and signatures to the client, which can notify the user that the records were found, but had been compromised (if the signature fails).

Administrators using BIND directly (without a GUI front end) will find that DNSSEC is extraordinarily difficult to deploy, and requires massive time and effort and extensive knowledge of PKI to produce signed resource records, even using the public domain PERL scripts. A good DNS appliance (like InfoWeapons SolidDNS) can totally automate both the signing and validation processes, which vastly speeds up signing an entire domain, eliminates many possible errors, and requires little or no security or cryptographic expertise on the part of the DNS administrator.

Ideally the private key used to sign records should be kept in an HSM (Hardware Storage Module), which the private key never leaves. The object to be signed is sent into the HSM, the private key is used to do the signing inside the HSM, and the result it retrieved and used. For highest security, the HSM should be certified by FIPS 140-2 or equivalent. FIPS is the U.S. Federal Information Processing Standards. FIPS 140-2 is the "Security Requirements for Cryptographic Modules", May 2001. Most commercially available HSMs are quite expensive, and their performance is very low compared to doing the same algorithm in software on modern processors (e.g. 1200 signatures per second for a very good HSM, as opposed to 12,000 signatures per second in software on entry level hardware). Typically an HSM is needed only on a *signing DNSSEC server*.

Digital signature verification (used in DNSSEC validation) uses only the *public key* corresponding to the private key used to create the signatures, so it can be done without an HSM, hence at much higher performance. The entry level model of SolidDNS can do about 30,000 queries per second without DNS verification, or about 27,000 per second with DNSSEC verification. Any DNS server can do DNSSEC validation, with no need for an HSM.

DNSSEC is not strictly an IPv6 technology, and is equally applicable to IPv4. It is however being deployed at the same time as IPv6, and of course, it is important that any system deployed to support DNSSEC be able to sign and validate both IPv4 and IPv6 related resource records, and can support queries over both IPv4 and IPv6. DNSSEC is a very important part of the Second Internet.

Essentially DNSSEC introduces *trust* into the DNS system.

Chapter 9 – IPv6 Related Organizations

There are quite a few international and national level organizations involved in making this transition from the First Internet to the Second Internet work. This chapter lists the most prominent ones, but does not claim to be comprehensive.

9.1 – Internet Governance Bodies

The first group of organizations helps govern the Internet. There is no Internet Corporation, or any U.N. Internet Authority. The Internet is something quite different from the kinds of entities most people are familiar with. Its ownership and management is as decentralized and trans-national as the physical implementation of the Internet itself. For example, what country is the Internet located in? All of them!

Anyone who really wants to can join one of these organizations, and the various groups address a variety of aspects of creating the standards that others use to build the hardware and software that make up the physical Internet. Other groups help manage resources, such as domain names, or Internet addresses. Others help resolve disputes and set policies that help the millions of owners of the various pieces of the Internet to get along and be willing to continue voluntarily connecting their networks to each other. Some national governments try to control or regulate the Internet, or the content on it, but the highly decentralized nature of it, and the difficulty of even pinning down what jurisdiction something on the Internet happened in, makes such control difficult at best. Any country whose people are forbidden access to the Internet are missing out on things that allow those who do have access to run circles around them competitively. It would be like a blind person and a fully sighted one having a sword fight, or a race between someone on foot, and someone in a Ferrari. It would all be over in seconds. China has had a very difficult time trying to maintain strict authoritarian communist rule, while enjoying the economic benefits of access to the Internet. They have tried to deploy what some call “the great firewall of China”, but there are many ways for people who understand the technology to gain access to those parts of it China doesn’t want their people to see. If they are really good they can do it without their government even being able to detect it.

So who *is* in charge? The easy answers are *no one* or *everyone*. It is possible to identify some organizations that clearly are in charge of some aspects of the Internet. Most of them are heavily involved in trying to help the users of the Internet survive a looming disaster (the depletion of the IPv4 address space), and migrate smoothly and safely to the wonderful new promised land, the Second Internet.

9.1.1 – Internet Corporation for Assigned Names and Numbers (ICANN)

ICANN was formed in 1998, as a not-for-profit public-benefit corporation. Participants all over the world help keep the Internet secure, stable and interoperable. ICANN does not try to control content on the Internet, stop spam or control access to the Internet. It does do the following things:

- Oversee the generic Top Level Domain (gTLD) names, and the country code Top Level Domain (ccTLD) names. They also oversee authorization of Internationalized Domain Names (IDNs) in various languages and scripts.
- Oversee operation of the DNS Root Servers.
- Draw up contracts with each Domain Name Registry.
- Oversee IANA (Internet Assigned Numbers Authority).
- Publish all corporate documents, bylaws, financial information, major agreements, policies, operating plan and strategic plan, at http://www.icann.org/en/documents/.
- Hold monthly meetings of the ICANN board to address issues and set policy. The minutes of each meeting are made publically available at http://www.icann.org/en/minutes/.

9.1.2 – Internet Assigned Numbers Authorities (IANA)

One of the key organizations with regards to both the old IPv4 addresses and the new IPv6 addresses is the Internet Assigned Numbers Authority (IANA). You can find their website at *www.iana.org*. They do the following things:

- Oversee the DNS Root Zone (creation and management of the generic TLDs and ccTLDs), as well as the *int* domain registry (for international organizations) and the *arpa* zone. The *arpa* TLD has several very important parts of DNS under it, such as the reverse zones for both IPv4 (*in-addr.arpa*) and IPv6 (*ip6.arpa*), plus the ENUM E.164 zones (*e164.arpa*). Some parts of these reverse hierarchies can be delegated to ISPs or even to end-user organizations, but IANA is in charge of the overall structure.
- Maintain the Interim Trust Anchor Repository (ITAR) for those parts of the Internet's domain space that has already been signed with DNSSEC. This is a temporary role until such time as the root of the entire domain space is signed. This is where you can find the public keys needed to verify DNSSEC signatures for signed DNS zones.
- Perform the top level management of IPv4 addresses. IANA allocates giant blocks of IPv4 addresses (called "/8s") with about 16.7 million addresses each, to the five Regional Internet Registries for the world, AfriNIC, APNIC, ARIN, LACNIC and RIPE NCC. As of this writing only 20 "/8" blocks are left to allocate (about 7.8% of the original 256). As we reach the "end days" for IPv4, the IANA will be the first to run out. They already have a plan for this. When they get down to five remaining "/8" blocks, they will allocate one of those to each of the five RIRs, and then close shop (as far as allocation of IPv4 addresses goes). This will probably happen on or before September 2011, by best current estimates. The RIRs will probably run out within six months after that. When they're gone, they're gone.
- Perform the top level management of IPv6 addresses. They perform the same basic allocation function with IPv6 addresses that they have done for many years with IPv4 address. The main difference is that there are a *lot* more IPv6 addresses. They allocate giant chunks of IPv6 addresses to the RIRs as needed. It is unlikely that IANA will ever run out, so long as there is something recognizably TCP/IP. There are enough IPv6 addresses just in the 2000::/3 block marked for allocation for every human alive today to get over 5,000 of the standard allocation blocks, which are "/48"s. Each "/48" is large enough for the biggest organization on earth.
- Manage AS numbers. AS stands for Autonomous Systems. It refers to complete networks at the top level of the routing hierarchy. Below the AS level, Interior Gateway Routing Protocols are

used (e.g. RIP2, EIGRP, etc.). At the AS level, Exterior Gateway Routing Protocols are used (e.g. BGP4 and BGP4+). Each Autonomous System network has a unique number. They have been using 16 bit numbers (which allowed 65,536 possible ASes). Just like with IPv4, we are running out of AS numbers, so they are in the process of changing to 32 bit AS numbers. That is causing some issues, but nothing like the issues related to changing from 32 bit IPv4 addresses to 128 bit IPv6 addresses. There is no worldwide "32 bit AS Number" Forum, or any need for one. The people affected are fairly savvy technically, and are simply making the changeover quietly.

- Allocate and assign IPv4 and IPv6 multicast addresses.
- Allocate and assign IPv6 anycast addresses.
- Allow people to reserve and register port numbers and other assigned numbers related to Internet protocols.

IANA is heavily involved in promoting the adoption of the IPv6 protocol throughout the world. They know how close they are to the bottom of the barrel with IPv4 addresses. They encourage the Regional Internet Registries to promote the adoption of IPv6, and each of them is doing this.

9.1.3 – Regional Internet Registries (RIRs)

There are five top level Registries directly below IANA, who set address allocation policy for their region and allocate blocks of both IPv4 and IPv6 addresses to ISPs and other interested parties. One way to obtain addresses is to join one of the registries and apply for addresses. Some regional registries charge for these, others provide them free. You can only obtain addresses from the registry in the region where you reside, or where the HQ of your organization is based.

Each Regional Internet Registry provides the following services for Internet users in their part of the world:

- IPv4 and IPv6 address space allocation, transfer and record maintenance
- Autonomous System number allocation, transfer and record maintenance
- Provide online directories of registration transaction information (WHOIS database)
- Provide online information about routing (Internet Routing Registry)
- Management of reverse DNS for addresses assigned by the RIR
- Hold periodic meetings and elections
- Perform education and training on relevant topics (such as IPv6)
- Maintain policy discussions on E-mail lists, conduct public policy meetings, and publish policy documents on their website

The three largest RIRs (ARIN, RIPE NCC and APNIC) are all aggressively advocating for adoption of IPv6. Like IANA, they know how many addresses are left, and how rapidly they are being allocated. They know that the "end times" for IPv4 allocation are near. All are strongly encouraging all ISPs and organizations that obtain addresses from them to begin adoption of IPv6 *now*. If the major oil companies told people that there was not going to be any gas for new cars made after a certain date (less than two years off), there would be a mad scramble to create and sell cars that ran on something else. This is just as big a deal, and according to OECD, will have very serious economic consequences for every country and organization that has not prepared for the end of IPv4 allocations.

The five Regional Internet Registries and their coverage areas are:

9.1.3.1 – American registry of Internet Numbers (ARIN) – www.arin.net

ARIN provides services to Internet users in North America (including the United States, Canada plus many Caribbean and North Atlantic islands).

ARIN runs an IPv6 Wiki at *www.getIPv6info.info*. This site includes book reviews, self education, IPv6 presentations and documents, survey results, planning information, management tools, etc.

On 7 May 2007, the ARIN Board of Trustees passed the following resolution:

> RESOLUTION OF THE BOARD OF TRUSTEES OF ARIN ON INTERNET PROTOCOL NUMBERING RESOURCE AVAILABILITY
>
> WHEREAS, community access to Internet Protocol (IP) numbering Resources has proved essential to the successful growth of the Internet; and,
>
> WHEREAS, ongoing community access to Internet Protocol version 4 (IPv4) numbering resources cannot be assured indefinitely; and,
>
> WHEREAS, Internet Protocol version 6 (IPv6) numbering resources are available and suitable for many Internet applications,
>
> BE IT RESOLVED, that this Board of Trustees hereby advises the Internet community that migration to IPv6 numbering resources is necessary for any applications which require ongoing availability from ARIN of contiguous IP numbering resources; and,
>
> BE IT ORDERED, that this Board of Trustees hereby directs ARIN staff to take any and all measures necessary to assure veracity of applications to ARIN for IPv4 numbering resources; and,
>
> BE IT RESOLVED, that this Board of Trustees hereby requests the ARIN Advisory Council to consider Internet Numbering Resource Policy changes advisable to encourage migration to IPv6 numbering resources where possible.
>
> Implementation of this resolution will include both internal and external components. Internally, ARIN will review its resource request procedures and continue to provide policy experience reports to the Advisory Council. Externally, ARIN will send progress announcements to the ARIN community as well as the wider technical audience, government agencies, and media outlets. ARIN will produce new documentation, from basic introductory fact sheets to FAQs on how this resolution will affect users in the region. ARIN will focus on IPv6 in many of its general outreach activities, such as speaking engagements, trade shows, and technical community meetings.

9.1.3.2 –Réseaux IP Européens Network Coordination Centre (RIPE NCC) – www.ripe.net

RIPE NCC provides services to Internet users in Europe, the Middle East and Central Asia. This includes:

- Southwest Asia: Azerbaijan, Bahrain, Cyprus, Georgia, Iran, Iraq, Israel, Jordan, Lebanon, Saudi Arabia, Syria, Turkey, UAE and Yemen
- Central Asia: Kazakhstan, Kyrgyzstan, Tajikistan, Turkmenistan and Uzbekistan
- Europe: Albania, Armenia, Austria, Belarus, Belgium, Bosnia-Herzegovina, Bulgaria, Croatia, Czech Republic, Denmark, Estonia, Finland, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Macedonia, Moldova, Montenegro, Norway, Netherlands, Poland, Romania, Russia, Serbia, Slovakia, Spain, Sweden, Switzerland, Turkey, Ukraine, United Kingdom and Yugoslavia
- North America: Greenland

RIPE NCC runs the "IPv6 Act Now" site (www.ipv6actnow.org) with lots of information on IPv6 for small businesses, enterprise, ISPs and government.

On October 26, 2007, RIPE NCC issued the following warning, which is included here, verbatim:

> During the RIPE 55 meeting in Amsterdam, the RIPE community agreed to issue the following statement on IPv4 depletion and the deployment of IPv6.
>
> "Growth and innovation on the Internet depends on the continued availability of IP address space. The remaining pool of unallocated IPv4 address space is likely to be fully allocated within two to four years. IPv6 provides the necessary address space for future growth. We therefore need to facilitate the wider deployment of IPv6 addresses.
>
> While the existing IPv4 Internet will continue to function as it currently does, the deployment of IPv6 is necessary for the development of future IP networks.
>
> The RIPE community has well-established, open and widely supported mechanisms for Internet resource management. The RIPE community is confident that its Policy Development Process meets and will continue to meet the needs of all Internet stakeholders through the period of IPv4 exhaustion and IPv6 deployment.
>
> We recommend that service providers make their services available over IPv6. We urge those who will need significant new address resources to deploy IPv6. We encourage governments to play their part in the deployment of IPv6 and in particular to ensure that all citizens will be able to participate in the future information society. We urge that the widespread deployment of IPv6 be made a high priority by all stakeholders."

RIPE NCC issued another warning concerning IPv4 address space depletion, on April 10, 2008:

> Currently, 180 of 256 blocks of "/8" have already been allocated. Of the remaining 76, 35 are already reserved for the Internet Engineering

Taskforce (IETF) and the remaining 41 blocks are held in the Internet Assigned Numbers Authority (IANA) pool for future allocation to the RIRs.

As IPv6 provides the necessary address space for future growth, RIPE NCC is urging business and government leaders to ease the path for wider deployment of IPv6 addresses. Failure to adopt these new resources could mean a slowing in the pace of Internet innovation.

“Now is the time to recognize that sustainable growth of the IPv4-based Internet is coming to an end, and that it is time to move on, with IPv6 ready as the successor.

“In order to sustain the impressive speed of Internet innovation and ensure a healthy Internet economy for the future, we recommend that content providers make their services available over IPv6,” comments Axel Pawlik, Managing Director at RIPE NCC.

“We view governments as key players in Internet growth and urge them to play their part in the deployment of IPv6 and in particular to lead by example in making content available in IPV6. Ultimately, we urge that the widespread deployment of IPv6 be made a high priority by all stakeholders."

When CIOs make firm decisions to deploy IPv6, the process is fairly straightforward. Staff will have to be trained, management tools will need to be enhanced, routers and operating systems will need to be updated, and IPv6-enabled versions of applications will need to be deployed. All these steps will take time.

The move to IPv6 will provide billions of further addresses through 128-bit addressing, which allows 50 billion, billion addresses for every person on the planet. Islands of IPv6 are already in use, but RIPE NCC argues that infrastructure support must be addressed in time for IPv6 to fulfill its predicted role as the catalyst for the next stage of Internet development.

Pawlik concludes: “We have well-established, open and widely supported mechanisms for Internet resource management and we’re confident that our Policy Development Process meets and will continue to meet the needs of all Internet stakeholders through the period of IPv4 exhaustion and IPv6 deployment. The immediate challenge lies in making content available in IPV6 using the processes and mechanisms already available to ensure that service providers and content providers build adequate experience and expertise in good time.”

Note that this warning was over two years ago, and at that time 41 “/8” blocks remained. As of August 2010, only 14 “/8” blocks remain unallocated.

9.1.3.3 – Asia Pacific Network Information Center (APNIC) – www.apnic.net

APNIC provides service to Internet users in:

- South Asia: Afghanistan, Bangladesh, Bhutan, India, Nepal, Pakistan and Sri Lanka
- Eastern Asia: China, North Korea, Hong Kong, Japan, Macau, Mongolia, South Korea and Taiwan
- Southeast Asia: Cambodia, Indonesia, Laos, Malaysia, Myanmar, Philippines, Singapore, Thailand and Vietnam
- Australia and New Zealand
- Oceania: various islands in Polynesia, Melanesia and Micronesia

APNIC is currently running a program called "Kickstart your IPv6", in which anyone that owns or obtains an IPv4 address allocation from APNIC can get a free block of IPv6 addresses. If their IPv4 block is less than a "/22", then the IPv6 block is a "/48". For IPv4 blocks from "/22" and up, the free IPv6 block is a "/32" (this is 4 billion times 4 billion times the size of the entire IPv4 address space). You could also look at this as 65,536 "/48" blocks. These addresses are not tied to any ISP, and can be routed from anywhere. There is no demonstration of need required for obtaining the IPv6 address block.

APNIC also runs an IPv6 resource site at *http://icons.apnic.net/display/ipv6/Home*. The name "icons" stands for *Internet Community of Online Networking Specialists*.

9.1.3.4 – Latin American and Caribbean Network Information Center (LACNIC) – www.lacnic.net

LACNIC was started in 2002. It provides services to Internet users in:

- North America: Mexico
- Central America: Costa Rica, El Salvador, Guatemala, Honduras, Nicaragua and Panama
- South America: Argentina, Belize, Bolivia, Brazil, Chile, Columbia, Ecuador, French Guiana, Paraguay, Peru, Uruguay and Venezuela
- Caribbean Islands: Aruba, Barbados, Cayman Islands, Cuba, Dominica, Dominican Republic, Grenada, Haiti, Jamaica, and various smaller islands

LACNIC runs an IPv6 resource site at *portalipv6.lacnic.net/en/portal-IPv6-2*

9.1.3.5. – Africa Region (AfriNIC) – www.afrinic.net

AfriNIC provides services for Internet users in the entire African continent. It began in April 2005.

They run an IPv6 resource center at *http://www.afrinic.net/IPv6*, and an IPv6 virtual lab at *www.afrinic.net/projects/cvl.htm*. This is a test network with public access, with primarily Cisco equipment.

9.1.4 – The Number Resources Organization (NRO) – www.nro.net

NRO was formed in October 2003 by the four Regional Internet Registries that existed at the time, to formalize their co-operative efforts. Its goal is to protect the unallocated Number Resource pool, to

promote and protect the bottom-up policy development process, and to act as a focal point for Internet community input into the RIR system. They run an IPv6 site at www.nro.net/ipv6.

Recently NRO issued the following statement, when the remaining IPv4 address pool dropped below 10%:

> "This is a key milestone in the growth and development of the global Internet," noted Axel Pawlik, Chairman of the NRO. "With less than 10 percent of the entire IPv4 address range still available for allocation to RIRs, it is vital that the Internet community take considered and determined action to ensure the global adoption of IPv6. The limited IPv4 addresses will not allow us enough resources to achieve the ambitions we all hold for global Internet access. The deployment of IPv6 is a key infrastructure development that will enable the network to support the billions of people and devices that will connect in the coming years," added Pawlik.

9.1.5 – Internet Architecture Board (IAB) – www.iab.org

"The IAB is chartered both as a committee of the Internet Engineering Task Force (IETF) and as an advisory body of the Internet Society (ISOC). Its responsibilities include architectural oversight of IETF activities, Internet Standards Process oversight and appeal, and the appointment of the RFC Editor. The IAB is also responsible for the management of the IETF protocol parameter registries."

9.1.6 –Internet Engineering Task Force (IETF) – www.ietf.org

"The mission of the IETF is to make the Internet work better by producing high quality, relevant technical documents that influence the way people design, use, and manage the Internet."

The IETF:

- Runs numerous working groups on technical topics relevant to the Internet, that are the main source of RFCs
- Oversees the standards process
- Maintains the Internet Drafts and the RFC Pages
- Holds periodic meetings (Fall, Spring and Summer, each year)
- Runs various mailing lists, which anyone can subscribe to

9.1.7 – Internet Research Task Force (IRTF) – www.irtf.org

"To promote research of importance to the evolution of the future Internet by creating focused, long-term and small Research Groups working on topics related to Internet protocols, applications, architecture and technology."

9.1.8 – Internet Society (ISOC) – www.isoc.org

"The Internet Society (ISOC) is a nonprofit organisation founded in 1992 to provide leadership in Internet related standards, education, and policy. With offices in Washington D.C., USA, and Geneva, Switzerland, it is dedicated to ensuring the open development, evolution and use of the Internet for the benefit of people throughout the world.

"The Internet Society provides leadership in addressing issues that confront the future of the Internet, and is the organisational home for the groups responsible for Internet infrastructure standards, including the Internet Engineering Task Force (IETF) and the Internet Architecture Board (IAB).

"The Internet Society acts not only as a global clearinghouse for Internet information and education but also as a facilitator and coordinator of Internet-related initiatives around the world. For over 15 years ISOC has run international network training programs for developing countries and these have played a vital role in setting up the Internet connections and networks in virtually every country connecting to the Internet during this time.

"The Internet Society has more than 100 organisational and more than 28,000 individual members in over 80 chapters around the world."

9.2 – IPv6 Forum Groups

There are a number of groups organized specifically to advocate for the adoption of IPv6, given the importance of the issue. There is an international umbrella group, called the IPv6 Forum, chaired by Latif Ladid, who wrote the foreword to this book. Their website is at *www.ipv6forum.org*.

9.2.1 – Local IPv6 Forum Chapters

There are local chapters of the IPv6 Forum in many countries. Some of these national groups use the term Forum (e.g. IPv6 Forum Downunder, at *www.ipv6forum.org.au*). Some use the term Task Force (e.g. North American IPv6 Task Force, at *www.nav6tf.org*). Some use the term *Council* (e.g. The German IPv6 Council, at *www.ipv6council.de*). Altogether there are currently 58 national or regional groups under the International IPv6 Forum. These groups advocate within their own country or region for the adoption of IPv6, and put on conferences usually called *IPv6 Summits*. There are links to all of the chapters on the IPv6 Forum international site (*www.ipv6forum.org*), as well as announcements about coming summits and other IPv6 related events.

9.2.2 – IPv6 Ready Logo Program

Affiliated with the IPv6 Forum is a group whose goal is to do testing of IPv6 equipment and applications, ISPs who offer IPv6, and websites that are available over IPv6. This testing and issuing of certifications is done under the IPv6 Ready Logo Program. Their website is at *www.ipv6ready.org*. There are three main parts to the IPv6 Ready Program: Products, ISP and website.

9.2.2.1 – IPv6 Ready Product Testing and Certification

Product Testing uses test suites developed by TAHI (part of the Japan WIDE project) and IPv6 ready Test Labs. This is overseen by the IPv6 Ready Logo Committee (v6LC). There are both Phase 1 ("Silver") tests, which verify behavior of the MUST clauses of all relevant RFCs, and Phase 2 ("Gold") tests, which verify behavior of both the MUST and the SHOULD clauses of all relevant RFCs. The hundreds of products that have passed these tests are published on the IPv6 Ready site at:

http://cf.v6pc.jp/logo_db/approved_list_ph1.php
http://cf.v6pc.jp/logo_db/approved_list_ph2.php

There are several categories of test suites currently. The IPv6 Ready Logo can be obtained for passing the *Core Protocols* tests, which includes both *Conformance* and an *Interoperability* tests. There are advanced tests in the following areas:

- IPsec – End-Node and Security Gateway
- IKEv2 – End-Node and Security Gateway
- Mobile IPv6 – Correspondent Node, Home Agent and Mobile Node
- NEMO – Home Agent and Mobile Router
- DHCPv6 – Client, Server and Relay Agent
- SIP – UA, Endpoint, B2BUA, Proxy, registrar
- SNMP – Management (SNMP-MIBs) – Agent and Manager
- MLDv2 – Multicast Listener Discovery protocol (version 2)

The current test sites include:

- BII: Beijing Internet Institute (Peoples Republic of China)
- CHT-TL: ChungHwa Telecom Labs (Taiwan)
- IRISA: Institut de Recherché en Informatique et Systemes Aleatories (European Union)
- IWC: InfoWeapons Corporation (Philippines)
- JATE: Japan Approvals Institute for Telecommunications Equipment (Japan)
- TTA: Telecommunication Technology Association (Korea)
- UNH-IOL: University of New Hampshire InterOp Lab (United States)

9.2.2.2 – IPv6 Enabled ISP and Website Certification

Information on how an ISP or a website can be certified as delivering IPv6 compliant service is available at:

www.ipv6forum.com/ipv6_enabled

The ISP certification process was created by the Beijing Internet Institute (BII). There is currently a basic level. The advanced level will be introduced shortly. The list of certified ISPs is available at:

www.ipv6forum.com/ipv6_enabled/isp/approval_list.php

Notably, Malaysia has taken this even further, and has three levels of ISP certification, which has been mandated by the Malaysian government.

- Phase 1 – Basic network connectivity tests
- Phase 2 – Interconnectivity tests
- Phase 3 – Commercial & advanced network services

12 ISPs have already passed the first two levels, and will shortly pass the third.

For websites, again there is a basic level currently available and an advanced level coming soon. The list of certified websites is available at:

http://www.ipv6forum.com/ipv6_enabled/approval_list.php

9.3 – Informal IPv6 Network Administration Certification

Hurricane Electric, in addition to providing free tunneled service via 6in4, 6to4 and Teredo, has put together an informal, self-administered certification program for IPv6 network administration. This covers aspects of IPv6 technology and implementation on various platforms (Linux, Windows, Cisco routers, etc.). There are several levels. At each succeeding level, you must answer harder questions. Several levels involve accomplishing actual network administration tasks, such as deploying an IPv6 compliant E-mail server. To obtain that level, you must exchange an E-mail with their IPv6 E-mail server. The site is available at *ipv6.he.net/certification/*. There are multiple levels, including Newbie, Explorer, Enthusiast, Administrator, Professional, Guru and Sage.

This is not a formal program, like Microsoft or Cisco certification, but it is free, and very educational. I have already qualified at the top level, as an IPv6 Sage. Here is my very cool certification badge:

9.4 – WIDE Project, Japan

Japan was an early leader in IPv6, and a consortium of Japanese IT companies headed by Professor Jun Murai has done some very important work that has significantly advanced the state of IPv6. These include reference implementations of the IPv6 stack for BSD (Kame) and Linux (USAGI), the test suites for the IPv6 Ready testing program (TAHI). They also provide IPv6 service to many developing countries in the region, such as the Philippines. Their website is available at *www.wide.ad.jp*.

The Kame project completed its task, and has been discontinued, but its website (www.kame.net) has been left up, complete with its famous turtle logo (kame is Japanese for turtle). If you connect over IPv4, the turtle just sits there. If you connect over IPv6, the turtle dances. In the early days of IPv6, it was a rite of passage to verify you had accomplished IPv6 connectivity by watching the Kame dance.

Chapter 10 – IPv6 Projects

There are a number of possible projects you can do for free, given the information in this book and various open source components (or evaluation versions of Microsoft products) readily available on the Internet.

It is possible to do the open source implementations based on FreeBSD, NetBSD, OpenBSD or various Linux flavors. Use the platform you are most familiar with. The BSD variants have a powerful dual stack packet filtering component called *pf*. This can be used to add a host-based firewall to any project (to block access via anything but the desired protocols), or even to build a multi-NIC router or firewall with complex rules. In Linux, the equivalent component is called NetFilter / IP Tables. The BSD and Linux packet filtering components have roughly the same functionality, but totally different deployment and configuration schemes. Both have one part that lives in the Kernel Space, and one part that lives in User Space. The configuration of the TCP/IPv4 and TCP/IPv6 stacks is done in different ways, but the functionality is almost the same. Both the BSD and Linux IPv6 implementations have passed IPv6 Ready Gold testing (at least one release, possibly not the most recent). For the most part, other open source components (Apache, Postfix, Dovecot, etc.) are pretty much the same regardless of what underlying platform is used.

Microsoft Windows Vista, Windows 7, Windows Server 2008 SP1 and Exchange 2007: each has excellent support for IPv6 and dual-stack operation. You can put together a viable testbed network with just Microsoft products if you like (except for the gateway router/firewall), or all with just open source, or mix and match. It all depends on your expertise and job requirements.

Some open source components (e.g. SMTP MTA, POP3/IMAP mail retrieval agents) are available in a variety of popular implementations (Postfix, QMail, EXIM, Dovecot, Cyrus IMAP, etc.). Pretty much all of these have support for IPv6, but in some cases, finding the specifics to actually deploy these in dual stack mode may be difficult to locate. I will recommend components that I have actually deployed and verified dual-stack operation, but if you happen to prefer a different component, chances are the necessary configuration information is available online somewhere.

If you are not looking for an educational experience, but just want to deploy working servers, you will be able to purchase “kits” from our website that have detailed instructions for deployment of the various servers listed in this projects chapter, using one possible set of open source components, along with configuration and testing procedures. This should help organizations (like universities) that are short on funds, but long on open source and network expertise to get started with IPv6 test bed networks. These are not recommended for mission critical production deployments in commercial organizations that require 24x7 support, high availability, simple GUI management, etc. There are good commercial products available from reputable companies that have all of these features and more. Even for such organizations, these kits may be enough (together with this book) to start climbing your own IPv6 learning curve and get some hands-on experience actually running real services dual stack.

Each project has a basic level of functionality described, and various extensions that can add more functionality (e.g. a basic router can be enhanced by adding packet filtering and/or proxies).

10.1 – Project 1: A Standalone Dual Stack Node in an IPv4 Network, using Tunneled Service

This is the simplest lab, and does not require a working dual-stack network. Each standalone node will be creating its own tunnel and the only IPv6 access to or from that node will be to or from the Second Internet via the tunnel. Local access will still be IPv4-only.

Freenet6 Tunneled Service using TSP

TSP tunnels (e.g. from Freenet6) will work even behind NAT, using NAT traversal with UDP encapsulation. If you do have a real IPv4 address (not behind any NAT), it will negotiate simple 6in4 tunneling. In either case, TSP requires client authentication (you must sign up at their site for credentials). For details, see *www.gogo6.com* and click on *Freenet6*. You can download their TSP client for a variety of platforms, including Windows, FreeBSD, Linux, etc. Follow their directions to download the appropriate TPC client, then install and configure it. This will typically even work from hotels. Some networks force your browser to use a web proxy, which usually is IPv4-only. In this case, web access will not work, but other protocols will typically work. If you can ping IPv6 addresses, but not surf to IPv6 websites, probably something is forcing your browser to use an IPv4-only web proxy. One good use for a standalone node is to test access to publically accessible servers in your testbed network from "outside".

As of this writing, Freenet6 includes their gogoclient in binary for Windows 32 bit, Windows 64 bit, and in source code for Linux, UNIX, Mac OS or BSD. Download the gogoclient User Guide and follow the directions.

Freenet6 has recently added DS-Lite virtual ISP service, plus a DS-Lite client, for Windows 32-bit and Windows 64-bit.

By default, with Freenet6 you get a single IPv6 address ("/128"), but you can request a "/64" or even a "/56" if your node can act as a gateway and route traffic to one or more subnets with multiple internal nodes.

Hurricane Electric 6in4 Tunneled Service

For Hurricane Electric, apply for an account at www.he.net (*tunnel broker* link). You will need a real IPv4 address (which accepts ping). This address cannot be behind NAT (even 1 to 1). Enter your IPv4 address as requested (e.g. *xxx.52.125.246*). The tunnel broker will create a tunnel and show you the details, for example:

```
Server IPv4 address:                    74.52.1094.74
Server IPv6 address:                    2001:470:1f04:b6d::1/64
Client IPv4 address:                    12.34.56.246
Client IPv6 address:                    2001:470:1f04:b6d::2/64

Anycasted IPv6 Caching Nameserver       2001:470:20::2
Anycasted IPv4 Caching Nameserver       74.82.42.42
```

```
Routed /48                              (click to allocate one)
Routed /64:                             2001:470:1f05:b6d::/64
```

With 6in4 tunneling, there is no client to download and install, the functionality is already in all supported platforms. All you need to do is configure it (as described below in labs 3 to 7 for various platforms).

By default, Hurricane Electric gives you a "/64" block but you can request a "/48" block if you have more than one subnet. Hurricane Electric's tunneling can be used to get IPv6 service for a single node, but more typically this would be used in a gateway node (router or firewall) to provide IPv6 service for one or more internal subnets.

Once the Tunnel is Set Up

Both of these virtual ISPs provide caching dual stack DNS service for client resolution, but you cannot add your own domain name or resource records into their DNS server. Hurricane Electric has recently added Free hosted dual-stack DNS service in addition to their caching DNS service for exactly this. In a later lab we will make use of this service.

Once the client is installed and configured, try pinging some IPv6 addresses (e.g. *ipv6.google.com* or *www.ipv6.org* – on FreeBSD or Linux remember to use *ping6*, not *ping*). If this works, try surfing to *www.ipv6.org* and see if it reports your IPv6 address. If surfing reports an IPv4 address, but pinging the IPv6 address works, likely your browser is being forced to use an outgoing IPv4-only proxy. This often happens in hotels.

On the BSD variants, the tunnel will appear as a pseudo interface named *gifx* (where x is a digit like 0). You can use ifconfig or other commands on pseudo interfaces just like real ones (such as *bge0*).

If you get a real IPv6 address on *www.ipv6.org*, welcome to the future! Start exploring the Second Internet.

10.1.1 – Standalone Node Lab 1: Freenet6 on Windows

Apply for an account at *www.gogo6.com* (Freenet6 tab), then download and install the TSP client for Windows 32-bit or Windows 64-bit (depending on your version of Windows). Download the gogoclient User Guide and follow directions.

10.1.2 – Standalone Node Lab 2: Freenet6 Using BSD or Linux

Apply for an account at www.gogo6.com(Freenet6 tab), then download the source code for the Linux, UNIX, Mac OS or BSD platforms, then build and install it (download and read the gogoclient User Guide for details).

10.1.3 – Standalone Node Lab 3: Hurricane Electric on Windows

Given the tunnel creation above, in Windows Vista, Windows 7 or Windows Server 2008, you would need to issue the following commands (substitute the addresses for your network and tunnel):

```
netsh interface teredo set state disabled
netsh interface IPv6 add v6v4tunnel IP6Tunnel 12.34.56.246 72.52.104.74
netsh interface IPv6 add address IP6Tunnel 2001:470:1f04:b6d::2
netsh interface IPv6 add route ::/0 IP6Tunnel 2001:470:1f04:b6d::1
```

10.1.4 – Standalone Node Lab 4: Hurricane Electric Using FreeBSD (since v4.4)

Given the tunnel creation above, you would need to issue the following commands (substitute the addresses for your network and tunnel):

```
ifconfig gif0 create
ifconfig gif0 tunnel 12.34.56.246 72.52.104.74
ifconfig gif0 inet6 2001:470:1f04:b6d::2 2001:470:1f04:b6d::1 prefixlen 128
route -n add -inet6 default 2001:470:1f04:b6d::1
ifconfig gif0 up
```

10.1.5 – Standalone Node Lab 5: Hurricane Electric on OpenBSD

Given the tunnel creation above, you would need to issue the following commands (substitute the addresses for your network and tunnel):

```
ifconfig gif0 tunnel 12.34.56.246 72.52.104.74
ifconfig gif0 inet6 alias 2001:470:1f04:b6d::2 2001:470:1f04:b6d::1 prefixlen 128
route -n add -inet6 default 2001:470:1f04:b6d::1
```

10.1.6 – Standalone Node Lab 6: Hurricane Electric on NetBSD / MacOS

Given the tunnel creation above, you would need to issue the following commands (substitute the addresses for your network and tunnel):

```
ifconfig gif0 create
ifconfig gif0 tunnel 12.34.56.246 72.52.104.74
ifconfig gif0 inet6 2001:470:1f04:b6d::2 2001:470:1f04:b6d::1 prefixlen 128
route -n add -inet6 default 2001:470:1f04:b6d::1
```

10.1.7 – Standalone Node Lab 7: Hurricane Electric Using Linux net-tools

Given the tunnel creation above, you would need to issue the following commands (substitute the addresses for your network and tunnel):

```
ifconfig sit0 up
ifconfig sit0 inet6 tunnel ::72.52.104.74
ifconfig sit1 up
ifconfig sit1 inet6 add 2001:470:1f04:b6d::2/64
route -A inet6 add ::/0 dev sit1
```

10.2 – Project 2: Dual Stack Router with Router Advertisement Daemon

This project is an ideal starting point for a test bed network. Any network with IPv6 clients requires a source of Router Advertisement messages, and either direct (or more likely, tunneled) IPv6 service. There are several sources of free tunneled IPv6 service today, including a large block of IPv6 addresses. There are tunnel clients available for each source of free tunneled service. You may wish to try more than one of the free services to see which works best for you, depending on where in the world you are deploying your test bed network.

It is possible to build a router starting with vanilla FreeBSD, OpenBSD or Linux. Each has built-in packet forwarding for both IPv4 and IPv6, and support for multiple NICs. Install the NICs before starting the operating system install, and tell it to make the system a *gateway*. This should enable the necessary packet forwarding, but you can always specifically enable it later. Details of doing this vary from one OS to another, search for "enable packet forwarding" together with your operating system name (e.g. "FreeBSD") in Google. All OSes also support creating static routes, default gateways, etc. The BSD variants have a powerful packet filtering component called *pf* that will allow you to turn your router into a firewall. Linux has similar components, called *Netfilter* and *IPTables*. The packet filtering components from both Operating Systems support IPv6. Creating a router (and even better a firewall) like this is highly educational (primarily about FreeBSD or Linux), and would make a great class project. However, if you are more interested in the end result than in the education, while still learning quite a bit about IPv6, there are several good alternatives.

One simple alternative is to buy a D-Link DIR-615 wireless router. It has support for dual stack, bringing in tunneled service, and IPv6 routing. It does not have any support for IPv6 packet filtering. There will be more and more such products hitting the market over the next two years. You may need to update the DIR-615 firmware to the latest version, as there is a lot of older stock in the channel that has pre-IPv6 firmware. The DIR-615 has supports both wireless clients and wired clients (there is a 4 port 10/100 switch included for wired clients, which you can chain out to a larger switch if needed).

Another alternative, somewhat more difficult, is to obtain one of various commercial SOHO routers or wireless routers for which open source replacement firmware is available. An example of this is the Linksys WRT-54G wireless router. Search google for "DD-WRT" for details. There are actually quite a few variants of the firmware for these devices, many of which support IPv6. Be very careful to research the exact hardware versions supported, as there are quite a few distinct devices sold under names like WRT-54G. This requires you to update the supplied vendor firmware with a completely different version, and the vendor will no longer provide support for your device. These units are typically available in the USD 50 to USD 100 price range, so this is an inexpensive way to get a dual stack gateway. There is quite a bit of information and support in online forums, but it is assumed that you have more technical knowledge and capability than a typical home user. Any of these units with replacement firmware will probably suffice, but may be somewhat limited in terms of tunnel and firewall rule support.

The third alternative is to create either a fairly complete dual stack firewall using m0n0wall. This is based on FreeBSD 6.2, but you download a bootable ISO image which installs everything, including a stripped down FreeBSD OS, all routing and packet filtering components, a web server, and a fairly sophisticated management GUI. This is what I recommend, as the result has quite good features and performance, but

it can still be created at very low cost. Their website is at *m0n0wall.ch* (note those are *zeroes* not *oh's*). There is quite a bit of information on their site on various options for hardware, and on configuration.

There are several competitors to m0n0wall, such as pfsense (also based on FreeBSD), and IPcop (based on Linux) but most do not currently include support for IPv6. Once you install one of these, you will not typically be interacting with the OS anyway (just using a point and click GUI), so it doesn't really matter which OS it is based on. m0n0wall uses the older FreeBSD packet filter (ipfw), while pfsense uses the newer packet filter (pf). Otherwise they are similar (except that pfsense does not currently support IPv6).

There are some fairly inexpensive "embedded" hardware platforms that can be used to deploy m0n0wall, such as ones from Soekris Engineering (*www.soekris.com*). m0n0wall even has downloadable images ready to put on a hard disk or even a Compact Flash card for various models of the Soekris box. This makes a highly portable, lightweight unit that is great for teaching, demos, etc. Since it includes various tunneling mechanisms, you can typically set it up to bring in IPv6 anywhere IPv4 service is available (hopefully including at least one "real" globally routable IPv4 address). Versions with anywhere from 3 to 7 NICs are available. If you don't want to hassle with a Soekris box, virtually any generic PC will work just fine, so long as it has at least two NICs. If performance is an issue, it is highly recommended that you use Intel Gigabit NICs, which are readily available. For typical Internet speeds, even an old decommissioned 486 or Pentium I/II/III box and 10/100 NICs will work just fine. If you are linking two or more gigabit networks with this gateway, use Intel gigabit NICs. There is also support for various wireless NICs, if you want to also support wireless dual-stack clients.

If this project is too complicated for you, you may wish to purchase a commercial dual stack router or firewall (such as InfoWeapons' *SolidWall*) and move on to the other projects. If you try to use a Cisco or Juniper router, be aware you may have to pay quite a bit extra for IPv6 support (in Cisco's case, you would need "Advanced IP Services" for IOS, which costs almost as much as the router). In a class situation, the instructor or lab assistants may wish to install and deploy this component for the students.

When you complete the labs in section 10.2, you will have made a great start on your complete Dual-Stack testbed network. We will add to that network in the following labs. Note that you will not be installing anything else on your router/firewall. You will need an additional internal node running FreeBSD for the remaining labs, configured for Dual-Stack operation.

10.2.1 – Router Lab 1: IPv4-only m0n0wall Installation and Configuration

This first lab creates a firewall that supports an IPv4-only LAN behind a single globally routable IPv4 address, with RFC 1918 private IPv4 addresses for internal nodes. We will add in support for bringing in tunneled IPv6 service in a later lab, making the network behind the firewall Dual-Stack.

Obtain a generic PC, possibly installing additional NICs (see comments above). You will need at least two NICs, although you can support as many as you have room for (for DMZ, etc.). m0n0wall only uses a very small amount of hard disk (<50MB), but it cannot partition the drive for multiple operating systems, so don't waste most of a nice 500GB drive on this. You should use a good quality drive, as the failure of an old (well used) drive will bring down your firewall. You can go to some extra trouble and put the firewall image in a Compact Flash card, but it will need something (floppy, USB thumb drive, etc.) to save its

configuration files onto. Using a CF card will make your firewall very reliable. The m0n0wall code will actually load into RAM and run there, to avoid burning out any bits in the CF card. Your PC should have a CDROM/DVD drive, but this can be provided using an external drive via USB (assuming your PC can boot from this). You will also need a video interface, a video display and a keyboard, but only during installation. Once m0n0wall is installed, these can be removed and your PC will operate as a "headless" node (no monitor, keyboard or mouse). All interaction with it is done via the network interface.

Step 1 – Get the m0n0wall installation media.

Download the "generic PC" ISO image (e.g. file *cdrom-1.32.iso*) from the *m0n0wall.ch* website, and burn it onto a writeable CDROM.

Step 2 – Setup your PC for installing m0n0wall.

Configure the ROM BIOS in your PC to not use ACPI, to work with an OS that does not support Plug and Play, and to boot from CD/DVD before booting from hard disk.

Step 3 – Boot from the m0n0wall install CDROM. When it finishes loading, you will see a menu:

```
LAN IP address: 192.168.1.1

Port configuration:

LAN -> sis0
WAN -> sis1

m0n0wall console setup
**********************

1) Interfaces: assign network ports
2) Set up LAN IP address
3) Reset webGUI password
4) Reset to factory defaults
5) Reboot system
6) Ping host
7) Install on Hard Drive
```

Choose option 7, *Install on Hard Drive*. It will list all available hard drives, and let you choose which one to use. For example:

```
Valid disks are:

       ad0    MAXTOR STM 3320620A 3.AAD   298.09 GB

Enter the device name you wish to install onto: ad0
```

It will warn you that this will overwrite the current contents of that drive.

```
Warning: ...

The firewall will reboot after installation.

Do you want to proceed? (y/n) y
```

When you type “y” (followed by Enter), it will write the firmware to the selected hard disk. When it is done, **remove the install CDROM** and allow the PC to reboot.

Step 4 –Install m0n0wall.

Connect a network cable from a powered-on switch to the m0n0wall NIC you want to use for LAN, so you can identify it in the next step (as being in the “up” state). It will display the console setup menu again (now minus option 7). For example:

```
LAN -> sis0
WAN -> sis1

m0n0wall console setup
**********************

1) Interfaces: assign network ports
2) Set up LAN IP address
3) Reset webGUI password
4) Reset to factory defaults
5) Reboot system
6) Ping host
```

Select option 1, *Interfaces: assign network ports*. It will list the available interfaces. In my case, the first NIC (vr0) was a D-Link PCI 10/100 PCI NIC I installed. The second NIC (vr1) was on the motherboard. The one named vr0 above is in the UP state, so that is the one to specify for “LAN”. Note: your interface names may differ from mine (the name depends on the FreeBSD driver used, which is determined by the NIC type). Do not install VLANs now.

```
Valid interfaces are:

vr0    00:15:e9:80:b3:b1 (up)     VIA VT6105 Rhine III 10/100baseTX
vr1    00:15:82:2e:b4:a5          VIA VT6102 Rhine II 10/100baseTX

Do you want to set up VLANs now? (y/n) n

Enter the LAN interface name or ‘a’ for auto-detection: vr0

Enter the WAN interface name or ‘a’ for auto-detection: vr1

Enter the Optional-1 interface name or ‘a’ for auto-detection:

The interfaces will be assigned as follows:

       LAN -> vr0
       WAN -> vr1

The firewall will reboot after saving changes
Do you want to proceed? (y/n) y

The firewall is rebooting now.
```

After it reboots, it will present the console setup menu again (now with your port assignments):

```
LAN IP address: 192.168.1.1
```

```
Port configuration:

LAN -> vr0
WAN -> vr1

m0n0wall console setup
**********************

1) Interfaces: assign network ports
2) Set up LAN IP address
3) Reset webGUI password
4) Reset to factory defaults
5) Reboot system
6) Ping host
```

Choose option 2: *Set up LAN IP address*. Follow the interaction below, replacing 192.168.1.1 with your preferred LAN IPv4 address, and the number of bits in the subnet mask (e.g. 24 for 255.255.255.0). Enable the DHCPv4 server, and provide it with a reasonable range of available IP addresses, in the same subnet as the LAN IP address. The remainder of the steps assume you used the default LAN IP address. For example:

```
Enter the new LAN IP address: 192.168.1.1
Enter the new LAN subnet bit count: 24
Do you want to enable the DHCP server on LAN (y/n) y
Enter start address of client address range: 192.168.1.101
Enter end address of client address range: 192.168.1.200

The LAN IP address has been set to 192.168.1.1/24
You can now access the webGUI by opening the following URL
in your browser

       http://192.168.1.1

Press ENTER to continue.
```

Step 5 – Connect remaining cables and configure client PC.

You are done with the installing m0n0wall, and will not need the video display and keyboard again. Remaining configuration is done from a browser on your client PC. Connect one network cable from the m0n0wall WAN port to your cable/DSL modem, and connect a second network cable from your client PC to the switch. Ideally your cable/DSL modem should be configured in the *bridge* mode (or be routing one or more real IPv4 addresses inside), so that you won't have multiple layers of NAT. Configure your client PC to obtain an IPv4 address automatically (with DHCPv4). Your client PC should acquire an IPv4 address somewhere in the range you specified for the DHCPv4 server. Try to ping the m0n0wall LAN interface from your client PC:

```
ping 192.168.1.1
```

Step 7 – Connect to m0n0wall with browser.

Assuming this worked, use the browser on your client PC to surf to *http://192.168.1.1*. You should see the m0n0wall login screen.

Log in with the default username (*admin*) and password (*mono*). You should now see the m0n0wall main screen.

Click on the *System* / *General Setup* link in the left column of that screen. Enter appropriate values for hostname (e.g. *m0n0wall*), domain (e.g. *hughesnet.local*), select a valid timezone (e.g. *Asia / Hong Kong*), and enter IPv4 addresses of your ISP's DNS servers (e.g. *4.2.2.2*, *4.2.2.3*).

Click the *Save* button at the bottom of the screen.

Step 8 – Configure WAN interface.

Click on the *Interfaces* / *WAN* link. Configure your WAN interface as appropriate for your ISP. In my case, I use a static globally routable IPv4 address of 12.34.56.120 with subnet mask length 26, my gateway is 12.34.56.129. (Note: the *12.34.56* is to hide my IP address). On the web form I entered:

```
Type:  Static

IP address:   12.34.56.120  / 26
Gateway:      12.34.56.129
```

Now click on the *Save* button at the bottom of the screen.

Step 9 – Verify interface status.

Click on the *Status / Interfaces* link. Both LAN and WAN interfaces should be *up*, and have the correct configuration. At this point there should not be any global unicast IPv6 addresses listed (if IPv6 is enabled on your client PC there may be some link local IPv6 addresses, which start with *fe80::*).

Step 10 – Test firewall.

Your client PC should have normal access to the IPv4 Internet at this point. Ping some internal addresses (e.g. *192.168.1.1*). Ping some external addresses (e.g. *4.2.2.2*). Ping some external nodes using symbolic names (e.g. *www.ipv6.org*). Finally, surf to an external node (e.g. *www.whatismyip.com*). You may wish to explore some of the IPv4 capabilities of m0n0wall at this point. By default, all outgoing IPv4 connections are allowed, and with *hide mode* NAT, there *are* no incoming IPv4 connections. The IPv6 menu options are not visible until you enable IPv6 in the next lab. You may want to connect some additional client nodes on the LAN side of the firewall. With the chosen private IP address range (*192.168.1.0/24*) and DHCPv4 configuration, you could connect another 100 or so nodes.

You could add some servers behind the firewall that accept connections from the outside world by using BINAT (also called 1:1 NAT), and an additional globally routable IPv4 address for each server. In future labs, we will configure this to allow incoming connections for Web and E-mail servers over both IPv4 and IPv6 for. You will discover that it is *much* easier to allow incoming connections over IPv6 than it is over IPv4.

10.2.2 – Router Lab 2: Adding IPv6 service using 6in4 Tunneling from Hurricane Electric

It is possible to also do this lab with 6to4 tunneling (which is also supported with m0n0wall), but I had less than satisfactory experience with it (especially with respect to Dual-Stack DNS), so I recommend that you use 6in4 tunneling to bring IPv6 service into your testbed network. The only other real candidate for tunneling is TSP, with service from gogo6. This will even work from behind a NAT gateway (no need for a routable IPv4 address). Unfortunately m0n0wall currently does not include support for TSP tunneling. You can build your own router/firewall from FreeBSD and use that if you like, but m0n0wall is much easier to work with.

There are actually quite a few sites that provide free 6in4 tunneled service that will work with m0n0wall (check out the list at SixXS) but this lab uses Hurricane Electric. We will request a "/64" block of addresses and use those to make the entire LAN behind m0n0wall Dual-Stack. We will also have m0n0wall provide Router Advertisements so that internal nodes can obtain valid IPv6 addresses through Stateless Address Autoconfiguration. No internal node needs to know anything about tunneling – inside the LAN, the subnet is *native* Dual-Stack.

You will need a "real" (globally routable) IPv4 address for the WAN interface of the firewall. This same address will be used for *hide mode* NAT for IPv4 (IPv4 NAT is configured automatically by m0n0wall). The firewall will need to be connected to the IPv4 Internet and responding to pings before you can create the 6in4 tunnel. Hurricane Electric has a number of PoPs (Points of Presence) around the world, and you should choose the one closest to you, to minimize latency.

Step 1 – Register for a Hurricane Electric account for creating and managing 6in4 tunnels.

> Surf to www.he.net and click on "Free IPv6 Tunnel Broker", or surf directly to *tunnelbroker.net*. Read the information provided, and then click on the *Register* button. Fill in the fields on the following screen. There can be only one account for a given E-mail address, but each account can request up to 5 tunnels. Each tunnel requires a distinct (not currently used for an HE tunnel), globally routable IPv4 address. Each tunnel can be a single "/64", or upgraded to a full "/48" (65,536 "/64" blocks) if desired. Since we will be working with two routing domains (see lab 10.2.3), you should upgrade your request to a full "/48".

Step 2 – Create your 6in4 tunnel.

> Once you are registered, or after you login, you will see a page summarizing any tunnels you have previously created, and "User Functions" to create new tunnels, etc. Click on *Create Regular Tunnel*. In the following screen, enter your globally routable IPv4 address in the box "IPv4 endpoint". This address must be live and "pingable" for this to work. The webpage will show what address you are connecting from, and choose the PoP closest to the source of that address. If you prefer to use another PoP for any reason, click the *Override* button and select any of the listed PoPs. Then click the *Submit* button.
>
> Upon successful creation of your tunnel, the details will be displayed. There is no need for a /48 in these labs. There are several important items listed in this form that you will need to complete the lab. If possible print this page for future reference.

Among other items, you will see (sample values from an old tunnel of mine are shown):

```
Server IPv4 address:                    216.218.221.6
Server IPv6 address:                    2001:470:1f05:c65::1/64
Client IPv4 address:                    12.34.56.160
Client IPv6 address:                    2001:470:1f05:c65::2/64

Available IPv6 Caching Nameserver:      2001:470:20::2
Anycasted IPv4 Caching Nameserver:      74.82.42.42

Routed /48:                             Allocate /48
Routed /64:                             2001:470:1f05:c65::/64
```

Step 3 – Configure firewall.

On your client computer, surf to the main configuration page for fw1 (192.168.1.1). Click on the *System / Advanced* link and click on the *Enable IPv6 support* link on the resulting screen to enable IPv6. Then click on the *Save* button immediately below that option.

Step 4 – Configure the WAN interface for tunneling.

Click on *Interfaces / WAN* link and enter the following information (use values appropriate for your tunnel), then click on the *Save* button. Do not enable Router advertisements on the WAN interface!

```
IPv6 mode           Tunnel                      (6in4 tunneling)

IPv6 address        2001:470:1f05:c65::2 / 64   (Client IPv6 address)

IPv6 tunnel endpt   216.218.221.6              (Server IPv4 address)
```

Step 5 – Configure LAN interface for IPv6 operation.

Click on the *Interfaces / LAN* link and enter the following information

```
IPv6 mode           Static

IPv6 address        2001:470:1f05:c65::1 / 64  (address from your /64)
```

Click on the *Send IPv6 router advertisements* link to enable this important function. This will make fw1 send periodic Router Advertisement messages with the routed IPv6 prefix (or reply to Router Solicitation messages from internal nodes). This is required for internal nodes to create IPv6 global unicast addresses. Later if you want to play with DHCPv6, you can worry about the following two options, but for now ignore them. You should also configure the above DNS servers in m0n0wall. Your client will do DNS resolutions using the LAN port of m0n0wall (using its *DNS forwarding* feature). Click on the *Save* button.

Step 6 – Allow desired IPv6 traffic to flow.

Unlike with IPv4, by default m0n0wall blocks *all incoming* and *all outgoing* IPv6 traffic. For starters, you might want to allow all outgoing IPv6 traffic and allow all incoming ICMPv6 traffic. You can refine this later. To do this, go to the *IPv6 Firewall Rules*, and create your first two rules (then click on "Apply Changes"), as follows:

Action	Interface	Protocol	Source	Destination
Pass	LAN	IPv6-ICMP any	any	LAN subnet
Pass	LAN	any	LAN subnet	any

Step 7 – Configure your internal client computer to support IPv6.

This is done with auto configuration both for the node address and for DNS (if you have issues with DNS, configure a real DNS server address). If you are using Windows, use the command "ipconfig /all" to verify that your node has obtained one or more valid IPv6 global unicast addresses. If you are using FreeBSD, use the command "ifconfig –a". The global unicast addresses will start with the "routed /64" prefix you entered, followed typically by a 64 bit *interface identifier* (probably random if your client computer is Windows Vista or Windows 7). You should also see a link-local IPv6 node address, and the link-local address of the gateway (in addition to the IPv4 default gateway).

Step 8 –Test network connectivity.

Ping some IPv6 nodes on the LAN, (e.g. *2001:470:1f05:c65::1*).

Ping some external IPv6 nodes, (e.g. *2001:470:20::2*).

Now ping some symbolic nodenames, such as *www.ipv6.org*.

Now surf to *www.ipv6.org*. The webpage there should indicate that you are connecting over IPv6, and the IPv6 address of your client computer. As one of the rites of passage, you must connect to *www.kame.net*, and watch the turtle dance (for many years, this has been the traditional proof that you have IPv6 working).

You may wish to connect some additional client computers behind your new firewall – you have enough global unicast IPv6 addresses for about 16 quadrillion of them! Of course most of those would have to be IPv6-only. In later labs, you will open some ports for incoming IPv6 traffic to specific nodes.

10.3 – Project 3: Internal Dual-Stack FreeBSD Server

In the first lab, we will first deploy FreeBSD 8.0 on a generic PC (IPv4-only) behind the m0n0wall just deployed. We allow outgoing IPv4 connections via any protocol, and incoming IPv4 connections via ping and ssh. We will then upgrade it to Dual-Stack using a fixed IPv6 address, and allow outgoing IPv6 connections via any protocol, and incoming IPv6 connections via ping and ssh. You will see that internal nodes are just as secure behind an IPv6 firewall with no NAT as they are behind an IPv4 firewall with NAT. The primary differences are that it is a *lot* easier to configure network access over IPv6, and many protocols that are broken by NAT now work fine. We will use this FreeBSD server in later labs.

You will need at least one NIC supported by FreeBSD (almost all are). This could be a built-in NIC on the motherboard, or a plug-in PCI NIC. We will not be installing X Windows or any window manager (KDE or Gnome). A GUI interface is not needed on a server, and merely wastes resources. Because we will be using a text-only display, virtually any graphics adapter will work. You will need a monitor and keyboard (but no mouse). You can install X and a window manager if you really want to (it will not prevent you doing any of the following labs), but this book does not cover how to do this. The memory and CPU requirements for FreeBSD are amazingly low compared to Windows Server 2008. Almost any PC you can find will run FreeBSD just fine. With a good recent machine and plenty of RAM, you can support many thousands of web or email users with no problem. This is an industrial grade operating system, with what many people consider the best TCP/IP implementation *anywhere*.

10.3.1 – FreeBSD Server Lab 1: IPv4-Only

Go to *www.freebsd.org* and download a copy of FreeBSD 7.3. If your CPU is an AMD 64 (or is an Intel processor that is "Intel 64" compliant), get the *amd64* (64-bit) version. If your CPU is an older Intel (not "Intel 64" compliant), get the *i386* (32-bit) version. You can look up your Intel processor on their website to find its capabilities, including whether it is "Intel 64" or not. Get the single disk DVD ISO image – it will simplify installation of packages later. If you are not familiar with FreeBSD, there is a lot of good information and tutorials online, and several good books listed in the bibliography. The *i386* version of FreeBSD will run fine on 64-bit machines, but at much lower performance than when running in native 64-bit mode.

In this lab we will install FreeBSD 7.3 on a generic PC, with support for IPv4 only. We will upgrade that to Dual-Stack in the next lab. There is extensive documentation available on the available options in the FreeBSD install. The recommended responses below will select options as needed for the following labs. Use them as specified unless you really know what you're doing. Note that any departures from my recommendations (except where noted) may cause problems (or changes to notes) in the following labs.

Step 1 – Configure your ROM BIOS for FreeBSD.

Select "OS does not support Plug and Play" in your ROM BIOS.

Set the boot priority to boot from CDROM/DVD before Hard Disk.

Insert the FreeBSD 7.3 Install DVD and reboot. You will shortly see the "Welcome to FreeBSD!" screen. You can hit Enter to continue, or just wait for 10 seconds, and it will continue by itself. The installer will probe your hardware and select appropriate drivers. This takes a few minutes – be patient.

Step 2 – Basic FreeBSD installation.

Select the recommended response (or an appropriate response for your case) to the following items. You can use up and down arrows on the keyboard, or tab key to highlight any of the options. Once the desired option is highlighted, pressing Enter will execute that selection. In many cases (e.g. *Yes*, *No*, *A for Auto defaults*, *Q to finish*), typing the first letter of the option will select and execute it immediately).

Country Selection

(Select the country you are based in, e.g. **PHILIPPINES**)

System Console Keymap

(Select the type of keyboard you are using. Most will select **USA ISO**.)

FreeBSD/amd64 7.3 – RELEASE – sysinstall Main Menu

Select **Standard – Begin a standard installation**

Message (about partitioning)

Select **OK**

User Confirmation Requested (about disk geometry)

"... Would you like to keep using the current geometry?"

Select **Yes**

FDISK Partition Editor

Select **A = Use Entire Disk**
Select **Q = Finish**

Install Boot Manager for drive ad0? (or whatever drive it finds)

Select **Standard – Install a standard MBR (no boot manager)**

Message (about BSD partitions)

Select **OK**

FreeBSD Disklabel Editor

Select **A = Auto Defaults**
Select **Q = Finish**

Choose Distributions

Select **Developer**

User Confirmation Requested

"Would you like to install the FreeBSD ports collection?"

Select **Yes**

Choose Distributions

Select **<<< X Exit**

Choose Installation Media

Select **1 – CD/DVD Install from a FreeBSD CD/DVD**

User Confirmation Requested

"Last Chance! Are you SURE you want to continue? ..."

Select **Yes**

(it will then create drive partitions, make file systems and copy files for 5-15 minutes)

Message – Congratulations!

"You now have FreeBSD installed on your system ..."

Select **OK**

User Confirmation Requested

"Would you like to configure any Ethernet or SLIP/PPP network devices?"

Select **Yes**

Network interface information required

(It will show a list of available network devices, some physical, some logical)

```
vr0    VIA VT3043/VT86C100A Rhine PCI Ethernet
plip0  <unknown network interface type>
sl0    SLIP interface on device /dev/cuad0 (COM1)
ppp0   PPP interface on device /dev/cuad0 (COM1)
```

(Highlight your physical network device, usually first listed)

Select **OK**

User Confirmation Requested

"Do you want to try IPv6 configuration of the interface?"

Select **No** (we will do this manually later)

User Confirmation Requested

"Do you want to try DHCP configuration of the interface?"

Select **No** (a server needs a fixed IPv4 address)

Interface Configuration

```
Host:              bsd1
Domain:            v6labs.com          (use your own domain name)
IPv4 Gateway:      192.168.1.1
Name Server:       4.2.2.2
IP address:        192.168.1.11
Netmask:           255.255.255.0
```

Select **OK**

User Configuration Requested

"Would you like to bring the vr0 interface up right now?"

Select **Yes**

User Configuration Requested

"Do you want this machine to function as a network gateway?"

Select **No**

User Configuration Requested

"Do you want to configure inetd and the network services that it provides?"

Select **No**

User Configuration Requested

"Do you want to enable ssh login?"

Select **Yes** (this will install sshd)

User Configuration Requested

"Do you want to have anonymous FTP access in this machine?"

Select **No**

User Configuration Requested

"Do you want to configure this machine as an NFS server?"

Select **No**

User Configuration Requested

"Do you want to configure this machine as an NFS client?"

Select **No**

User Configuration Requested

"Do you want to customize your system console settings?"

Select **No**

User Configuration Requested

"Would you like to set this machine's time zone now?"

Select **Yes**

Select local or UTC (Greenwich Mean Team) clock

"Is this machine's CMOS clock set to UTC ..."

Select **No**

Time Zone Selection

"Select a region"

Select your region (e.g. **Asia**)

Countries in <selected region>

"Select a Country or Region"

Select your country (e.g. **Philippines**)

Confirmation

"Does the abbreviation xxx (e.g. PHT) look reasonable?"

Select **Yes**

User Configuration Requested

"Does the system have a PS/2, serial or bus mouse?"

Select **No** (we do not need a mouse for a text based console)

User Configuration Requested

"Do you want to browse FreeBSD ports collection now?"

Select **Yes**

Select **Security**, then **sudo-1.7.2.5**

Select **Shells** then **pdksh-5.2.14p2_2**

Select **Install**

Package Targets

(It will list selected packages, e.g. sudo, pdksh)

Select **OK**

Adding packages

(the selected packages will install from the DVD, which takes a few minutes)

User Configuration Requested

"Would you like to add any initial user accounts to the system?"

Select **Yes**

User and group management

Select **Group**

Select **OK**

Add a new group

Group name: **admin**
GID: **1001**

Select **OK**

Select **User**

Select **OK**

Add a new user

Login ID: **admin**
UID: **1001**
Group: **admin**
Password: **admin$pw** (specify your own admin password here)
Full Name: **System Administrator**
Member Groups: **wheel** (enable admin to use *sudo* command)
Home directory: **/home/admin**
Login shell: **/usr/local/bin/ksh** (or use default shell)

Select **OK**

Select **X – Exit**

Message

"Now you must set the system manager's password ..."

Select **OK**

New Password: **root$pw** (specify your root password, chars don't echo)
Retype New Password: **root$pw** (repeat new root password, chars don't echo)

User Configuration Requested

"Visit the general configuration menu for a chance to set any last options?"

Select **No**

Select **X – Exit Install**

"Are you sure you wish to exit? The system will reboot (be sure to remove any floppies/CD/DVDs from the drives)"

```
Select    YES
```

(the system will now reboot – **remove the install DVD** before it starts loading!)

Step 3 – Reboot computer and log in.

The FreeBSD startup scripts will run, then the system ID is displayed, followed by the login prompt:

```
FreeBSD/amd64 (bsd1.v6lab.com) (ttyv0)

login: admin
password: admin$pw         (note: characters typed will not echo)

Last login: Tue May 18 23:50:04 2010
Copyright (c) 1980, 1983, 1986, 1988, 1990, 1991, 1993, 1994
        The Regents of the University of California.  All rights reserved.

FreeBSD 7.3-RELEASE (GENERIC) #0: Sun Mar 21 05:25:24 UTC 2010

<the Message Of The Day will display – you can edit this in /etc/motd>

$
```

FreeBSD is now up and running, and you are logged in as user *admin*. The "$" is the default Korn Shell *command prompt*. That means FreeBSD is waiting for you to enter a command. To do things requiring root privilege, enter the command **su**, followed by the root password (e.g. **root$pw**). The command prompt will change to "**#**" when you have root privilege. To exit root privilege, enter the command **exit**. For details on FreeBSD commands, check online, or in any of the books on FreeBSD listed in the Bibliography. You should learn at least one screen-oriented text editor, either *vi* (the default UNIX screen oriented text editor) or *edit* (a simple emacs-like editor). Again, there is documentation on these online and in books (edit is a simple emacs style editor). I happen to like the *uemacs* text editor (also similar to emacs, but a little more complete), which can be installed from the FreeBSD ports collection (go to directory */usr/ports/editors/uemacs* and build it).

Step 4 – View and test network configuration.

Use "ifconfig –a" to view current network configuration of all interfaces (vr0 is physical Ethernet interface, lo0 is the "loopback" logical interface). Ping the FreeBSD node itself (*192.168.1.11*), the default gateway (*192.168.1.1*), an external node (*4.2.2.2*) and an external node using symbolic nodename (*www.ipv6.org*). Note that ping will keep running until you stop it by typing Ctrl-C.

```
$ ifconfig –a
vr0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500
        options=2808<VLAN_MTU,WOL_UCAST,WOL_MAGIC>
        ether 00:15:f2:2e:b4:a5
        inet 192.168.1.11 netmask 0xffffff00 broadcast 192.168.1.255
        media: Ethernet autoselect (100baseTX <full-duplex>)
        status: active
plip0: flags=108810<POINTOPOINT,SIMPLEX,MULTICAST,NEEDSGIANT> metric 0 mtu 1500
```

```
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> metric 0 mtu 16384
        inet6 fe80::1%lo0 prefixlen 64 scopeid 0x3
        inet6 ::1 prefixlen 128
        inet 127.0.0.1 netmask 0xff000000

$ ping 192.168.1.11
PING 192.168.1.11 (192.168.1.11): 56 data bytes
64 bytes from 192.168.1.11: icmp_seq=0 ttl=64 time=0.041 ms
64 bytes from 192.168.1.11: icmp_seq=1 ttl=64 time=0.008 ms
^C
--- 192.168.1.11 ping statistics ---
3 packets transmitted, 3 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.008/0.019/0.041/0.016 ms

$ ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1): 56 data bytes
64 bytes from 192.168.1.1: icmp_seq=0 ttl=64 time=0.600 ms
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=0.467 ms
^C
--- 192.168.1.1 ping statistics ---
2 packets transmitted, 2 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.467/0.533/0.600/0.067 ms

$ ping 4.2.2.2
PING 4.2.2.2 (4.2.2.2): 56 data bytes
64 bytes from 4.2.2.2: icmp_seq=0 ttl=52 time=203.961 ms
64 bytes from 4.2.2.2: icmp_seq=1 ttl=52 time=205.177 ms
^C
--- 4.2.2.2 ping statistics ---
2 packets transmitted, 2 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 203.961/204.569/205.177/0.608 ms

$ ping www.ipv6.org
PING shake.stacken.kth.se (130.237.234.40): 56 data bytes
64 bytes from 130.237.234.40: icmp_seq=0 ttl=42 time=408.089 ms
64 bytes from 130.237.234.40: icmp_seq=1 ttl=42 time=413.438 ms
^C
--- shake.stacken.kth.se ping statistics ---
3 packets transmitted, 2 packets received, 33.3% packet loss
round-trip min/avg/max/stddev = 408.089/410.764/413.438/2.674 ms

$
```

Step 5 – Use *netstat* to see what ports are being listened to:

```
# netstat -na
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
tcp4       0     52 192.168.1.11.22        192.168.1.100.50057    ESTABLISHED
tcp4       0      0 *.22                   *.*                    LISTEN
udp4       0      0 *.514                  *.*
```

At this time, ssh (port 22/tcp) and syslog (port 514/udp) are listening only on IPv4.

Step 5 – Install *putty* on your Windows client node.

This will be used to connect to the FreeBSD server using the ssh (“secure shell”) protocol.

Download the putty open source ssh client for Windows (*www.putty.org*, among other places). Install it on your Windows client connected to NET1. Use it to connect to the new FreeBSD node by IP address (192.168.1.11). The first time you connect, it will warn you that this is an unrecognized system – accept the key and continue. Login as *admin* with password *admin$pw*. Note that you cannot login as *root* directly for security reasons. This is one reason to create a non-root account (admin) with the ability to elevate to root privilege using the *su* command when needed. You can open any number of ssh and/or telnet connections to your FreeBSD server from any number of other computers with network connectivity. The ssh protocol creates an encrypted tunnel, while telnet is in plaintext. The ssh server (sshd) is installed and automatically started at boot time if you select in during FreeBSD installation (or have installed it from the ports collection).

All remaining labs can be done from your Windows PC using *putty* or from the FreeBSD console, as desired. Connections over ssh from outside the firewall (to test access from outside) will require some kind of ssh client (e.g. putty), on a node that is located outside of your firewall.

Step 6 – Allow access via ssh and ping from outside of the firewall over IPv4.

This requires a second globally routable IPv4 address, setting up 1:1 NAT in m0n0wall between the second routable IPv4 address and bsd1's internal IPv4 address, adding a *proxy arp*, and adding some new IPv4 firewall rules on m0n0wall. This is how you make an IPv4 server accessible from outside your network today (because of the presence of NAT). This process is far simpler with IPv6, as you will see in the next lab. Note: if you haven't got a second routable IPv4 address, you can get by with using a single routable IPv4 address, by using m0n0wall's INBOUND NAT (this is similar to *port redirection* on other firewalls).

Create 1:1 NAT mapping: Click on the *Firewall / NAT* link. Click on the *1:1* tab on the NAT screen. Enter your second routable IPv4 address (e.g. *12.34.56.161/32*) as the *External IP address* and the internal address of bsd1 (*192.168.1.11/32*) as *the Internal IP address*. Allow m0n0wall to automatically create the necessary *proxy arp*. Verify the created proxy arp by clicking on *Services / Proxy ARP*. It should show Interface *WAN*, Network *12.34.56.161* (your second routable IPv4 address).

Create the necessary IPv4 firewall rules: Click on the *Firewall / Rules* link. No new LAN rules are required. On the WAN tab, enter two new rules:

```
Action = Pass, Proto = TCP, Source = any, Port = any,
Destination = 192.168.1.11, Port = 22

Action = Pass, Proto = ICMP (any), Source = any, Port = any,
Destination = LAN net, Port = any
```

Click on the *Apply Changes* button.

Now, from a node outside of your LAN (possibly via ssh to an external FreeBSD node, or ask a friend somewhere to do this for you), ping the external IP address of bsd1 (e.g. ping 12.34.56.161), then connect to it via ssh.

```
$ ping 12.34.56.161
PING 12.34.56.161 (12.34.56.161): 56 data bytes
64 bytes from 12.34.56.161: icmp_seq=0 ttl=64 time=0.043 ms
64 bytes from 12.34.56.161: icmp_seq=1 ttl=64 time=0.010 ms
^C
--- 12.34.56.161 ping statistics ---
2 packets transmitted, 2 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.010/0.026/0.043/0.016 ms

$ ssh admin@12.34.56.161
The authenticity of host '12.34.56.161 (12.34.56.161)' can't be established.
DSA key fingerprint is e3:b6:44:e6:10:60:46:b3:75:2e:ec:b8:3f:f1:6c:2d.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '12.34.56.161' (DSA) to the list of known hosts.
Password: admin$pw        (chars do not echo)
Last login: Thu May 20 16:55:26 2010 from 2001:418:5403:2
Copyright (c) 1980, 1983, 1986, 1988, 1990, 1991, 1993, 1994
        The Regents of the University of California.  All rights reserved.

FreeBSD 7.3-RELEASE (GENERIC) #0: Sun Mar 21 05:25:24 UTC 2010

$ uname -a
FreeBSD bsd1.v6lab.com 7.3-RELEASE FreeBSD 7.3-RELEASE #0: Sun Mar 21 05:25:24 UTC
2010     root@driscoll.cse.buffalo.edu:/usr/obj/usr/src/sys/GENERIC  amd64

$ exit
Connection to 12.34.56.161 closed.
$
```

Your new FreeBSD server is now available from anywhere on the IPv4 Internet! But before you panic, this access is controlled by the IPv4 rules in the firewall.

10.3.2 – FreeBSD Server Lab 2: Add Support for IPv6

Step 1 – Login with root privilege.

Login as *admin*, password *admin$pw*. Elevate privilege to root with *su* command, password *root$pw*. In the following, replace the string "vr0" with the interface name from your FreeBSD install (as in "IPv6_ifconfig_vr0"). Edit the file /etc/rc.conf, add the following lines:

```
IPv6_enable="YES"
IPv6_ifconfig_vr0="2001:470:1f05:c65::11 prefixlen 64"
IPv6_defaultrouter="2001:470:1f05:c65::1"
```

Step 2 – Edit the file /etc/hosts

Add the following lines:

```
2001:470:1f05:c65::11     bsd1.v6lab.com  bsd1
2001:470:1f05:c65::11     bsd1.v6lab.com.
```

Step 3 – Reboot the FreeBSD node and log back in with root privilege.

```
# reboot
...
```

```
login as: admin
Password: admin$pw
$ su
password: root$pw
#
```

Step 4 – Use *netstat* to see what ports are being listened to:

```
# netstat -na
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
tcp4       0     52 192.168.1.11.22        192.168.1.100.50057    ESTABLISHED
tcp4       0      0 *.22                   *.*                    LISTEN
tcp6       0      0 *.22                   *.*                    LISTEN
udp4       0      0 *.514                  *.*
udp6       0      0 *.514                  *.*
```

At this time, ssh (port 22/tcp) and syslog (port 514/udp) are running dual-stack.

Step 5 – Verify configuration and test IPv6 connectivity:

```
bsd1# ifconfig -a
vr0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500
        options=2808<VLAN_MTU,WOL_UCAST,WOL_MAGIC>
        ether 00:15:f2:2e:b4:a5
        inet6 fe80::215:f2ff:fe2e:b4a5%vr0 prefixlen 64 scopeid 0x1
        inet 192.168.1.11 netmask 0xffffff00 broadcast 192.168.1.255
        inet6 2001:470:1f05:c65::11 prefixlen 64
        media: Ethernet autoselect (100baseTX <full-duplex>)
        status: active
plip0: flags=108810<POINTOPOINT,SIMPLEX,MULTICAST,NEEDSGIANT> metric 0 mtu 1500
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> metric 0 mtu 16384
        inet6 ::1 prefixlen 128
        inet6 fe80::1%lo0 prefixlen 64 scopeid 0x3
        inet 127.0.0.1 netmask 0xff000000

bsd1# ping6 2001:470:1f05:c65::11
PING6(56=40+8+8 bytes) 2001:470:1f05:c65::11 --> 2001:470:1f05:c65::11
16 bytes from 2001:470:1f05:c65::11, icmp_seq=0 hlim=64 time=0.093 ms
16 bytes from 2001:470:1f05:c65::11, icmp_seq=1 hlim=64 time=0.028 ms
^C
--- 2001:470:1f05:c65::11 ping6 statistics ---
2 packets transmitted, 2 packets received, 0.0% packet loss
round-trip min/avg/max/std-dev = 0.028/0.060/0.093/0.033 ms

bsd1# ping6 2001:470:1f05:c65::1
PING6(56=40+8+8 bytes) 2001:470:1f05:c65::11 --> 2001:470:1f05:c65::1
16 bytes from 2001:470:1f05:c65::1, icmp_seq=0 hlim=64 time=1.409 ms
16 bytes from 2001:470:1f05:c65::1, icmp_seq=1 hlim=64 time=0.544 ms
^C
--- 2001:470:1f05:c65::1 ping6 statistics ---
2 packets transmitted, 2 packets received, 0.0% packet loss
round-trip min/avg/max/std-dev = 0.544/0.977/1.409/0.433 ms

bsd1# ping6 2001:470:20::2
PING6(56=40+8+8 bytes) 2001:470:1f05:c65::11 --> 2001:470:20::2
16 bytes from 2001:470:20::2, icmp_seq=1 hlim=63 time=87.920 ms
16 bytes from 2001:470:20::2, icmp_seq=2 hlim=63 time=207.857 ms
^C
--- 2001:470:20::2 ping6 statistics ---
```

```
3 packets transmitted, 2 packets received, 33.3% packet loss
round-trip min/avg/max/std-dev = 87.920/147.888/207.857/59.969 ms

bsd1# ping6 www.ipv6.org
PING6(56=40+8+8 bytes) 2001:470:1f05:c65::11 --> 2001:6b0:1:ea:202:a5ff:fecd:13a6
16 bytes from 2001:6b0:1:ea:202:a5ff:fecd:13a6, icmp_seq=0 hlim=48 time=557.031 ms
16 bytes from 2001:6b0:1:ea:202:a5ff:fecd:13a6, icmp_seq=1 hlim=48 time=529.053 ms
^C
--- shake.stacken.kth.se ping6 statistics ---
2 packets transmitted, 2 packets received, 0.0% packet loss
round-trip min/avg/max/std-dev = 529.053/543.042/557.031/13.989 ms

bsd1#
```

Step 6 – Allow access to bsd1 via IPv6 from outside your LAN.

Your BSD node's global unicast IPv6 address (e.g. 2001:470:1f05:c65::11) is already globally routable, so there is no need for 1:1 NAT or a proxy arp. All you need to do is add some IPv6 firewall rules. Click on the *Firewall / IPv6 Rules* link. Click on the LAN tab. Add two new rules (remember all incoming and outgoing IPv6 traffic are blocked by default):

```
Action = Pass, Proto = IPv6-ICMP, Source = any, Port = any,
Destination = LAN subnet, Port = any

Action = Pass, Proto = any, Source = LAN subnet, Port = any,
Destination = any, Port = any
```

These allow incoming ICMPv6, and all outgoing protocols. Click on the *Apply Changes* button.

Now click on the WAN tab. Add two more rules:

```
Action = Pass, Proto = IPv6-ICMP, Source = any, Port = any,
Destination = any, Port =any

Action = Pass, Proto = TCP, Source = any, Port = any,
Destination = 2001:470:1f05:c65::11, Port = 22
```

These allow incoming ICMPv6, and incoming ssh (port 22) over IPv6. Click on the *Apply Changes* button.

Now, from a node outside of your LAN (possibly via ssh to an external FreeBSD node, or ask a friend somewhere *who has IPv6* to do this for you), ping the IPv6 address of bsd1 then connect to it via ssh.

```
$ ping6 2001:470:1f05:c65::11
PING6(56=40+8+8 bytes) 2001:470:1f05:c65::11 --> 2001:470:1f05:c65::11
16 bytes from 2001:470:1f05:c65::11, icmp_seq=0 hlim=64 time=0.097 ms
16 bytes from 2001:470:1f05:c65::11, icmp_seq=1 hlim=64 time=0.037 ms
^C
--- 2001:470:1f05:c65::11 ping6 statistics ---
2 packets transmitted, 2 packets received, 0.0% packet loss
round-trip min/avg/max/std-dev = 0.037/0.067/0.097/0.030 ms

$ ssh admin@2001:470:1f05:c65::11
```

```
The authenticity of host '2001:470:1f05:c65::11 (2001:470:1f05:c65::11)' can't be
established.
DSA key fingerprint is e3:b6:44:e6:10:60:46:b3:75:2e:ec:b8:3f:f1:6c:2d.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '2001:470:1f05:c65::11' (DSA) to the list of known hosts.
Password:
Last login: Thu May 20 17:27:02 2010 from 12.34.56.161
Copyright (c) 1980, 1983, 1986, 1988, 1990, 1991, 1993, 1994
        The Regents of the University of California.  All rights reserved.

FreeBSD 7.3-RELEASE (GENERIC) #0: Sun Mar 21 05:25:24 UTC 2010

Welcome to FreeBSD!

$ uname -a
FreeBSD bsd1.v6lab.com 7.3-RELEASE FreeBSD 7.3-RELEASE #0: Sun Mar 21 05:25:24 UTC
2010     root@driscoll.cse.buffalo.edu:/usr/obj/usr/src/sys/GENERIC  amd64

$ exit
Connection to 2001:470:1f05:c65::11 closed.
$
```

Your new FreeBSD server can now connect to, and is available from anywhere on the IPv6 Internet! But before you panic, remember that this is subject to the rules in your firewall. By default, the m0n0wall (and this will be typical of IPv6 firewalls) will block at least all incoming connections, and in many cases all outgoing connections. You will need to specifically allow any incoming IPv6 connections, to specific internal nodes (by address) and via specific protocols (by port). Your internal nodes are just as safe from unwanted incoming connections as IPv4 nodes are behind "hide mode NAT", but *wanted* incoming connections are far easier to allow.

10.3.3 – FreeBSD Server Lab 3: Install Gnome GUI for FreeBSD (optional)

A FreeBSD server does not really need a GUI interface, and many system administrators consider one to be unnecessary overhead on a server. However, if you prefer, you can add the KDE or Gnome GUI to FreeBSD to make it more like a Windows Server. This will allow you to test later labs from the included browser and E-mail client rather than using a Windows node. This lab installs Gnome, one of the more popular GUIs for UNIX-like systems. If you prefer KDE, see information online or in books regarding installation and configuration of that.

Step 1 – Install the Gnome packages.

> Insert the FreeBSD 7.3 Installation DVD in the drive of your bsd1 server. Login as root (username *root*, password *root$pw*). Type the command *sysinstall*. In the resulting menu system, select *Configure*, then *Packages*, then select installation medium as CD/DVD. In the list of package categories, select *gnome*. In the list of Gnome related packages, highlight *gnome2-2.28.2_1* by moving to it with arrow keys then hitting the space bar. Hit the *tab* key to move to the OK button. Hit the Enter key. Tab to the Install options and hit the Enter key. This will take a while as there are many parts to Gnome. This will not install Open Office, but you can add that using sysinstall if you wish. When it is done, exit from sysinstall.

Step 2 – Enable procfs.

For some reason, FreeBSD does not do this automatically when you install Gnome.

Edit the file */etc/fstab*. Add the following information at the end:

```
proc   /proc  procfs rw     0      0
```

Step 3 – Arrange for Gnome to start automatically at boot time.

For the fully automated Gnome startup, edit the file */etc/rc.conf* and add the following lines (the moused_enable line may already be there).

```
moused_enable="YES"
gnome_enable="YES"
```

Reboot your system. The GUI (complete with *gdm* login manager) will come up automatically after the reboot. The available users will be listed in the *gdm* User List. Choose one and enter the password. Note that *root* will not appear as one of the available users. You can still login as *root* by selecting *other* then entering username root. You can of course login as *admin* and use *su* to elevate to root privilege, in a Console terminal. There are also ways to allow manual startup of X and Gnome without having gdm control login. Details can be found in online guides and some books on FreeBSD.

If you ever want to go back to a text-only FreeBSD system, comment out the line in file */etc/rc.conf* with "gnone_enable" (change it to *#gnome_enable="YES"*), and reboot.

10.4 – Project 4: Dual Stack DNS Server

Any dual-stack network requires dual-stack DNS service. You should obtain a test domain from a domain registrar that supports registering the NS records in the TLD servers with both IPv4 and IPv6 addresses, such as godaddy.com. The names and addresses of all internal nodes should be populated in a DNS authoritative zone for your test domain. You can also publish the names and addresses of nodes for externally visible nodes in another BIND "view".

Assignment of IPv4 node addresses and other network configuration (default gateway, subnet mask, IPv4 addresses of DNS servers) can be done manually on each node, or automated with DHCPv4.

Assignment of IPv6 node addresses can be done manually, or more likely with Stateless Address Auto-configuration (which requires a source of Router Advertisement messages on the network). The link-local address of the default gateway typically is done automatically via Neighbor Discovery protocol (assuming there is a dual stack gateway on the network). The IPv6 addresses of DNS (and other) servers can be done manually, or automated via stateless DHCPv6. With stateful DHCPv6, it is possible to assign IPv6 addresses automatically in a more controlled manner.

You can do this using Open Source components, or with Windows Server 2008, or both (but only one DHCPv4 server should be active on any given network).

If this project is too complicated, you can contact InfoWeapons to use our free trial hosted dual-stack DNS service (for a limited number of nodes), or sign up for a commercial hosted account. This is implemented with our SolidDNS appliance with intuitive GUI administration. You will get an "organization account" which allows you to see and modify only your own domain information. You can also purchase one or more InfoWeapons' SolidDNS appliances to provide dual stack DNS (with DNSSEC and ENUM), plus DHCPv4 and DHCPv6 service (stateless and stateful) in your testbed network.

Another alternative is to deploy Microsoft Windows Server 2008 SP1 as another node behind the m0n0wall. It includes a fairly complete dual stack DNS server (as well as DHCPv4 and DHCPv6 servers). These are simple to manage via GUI administrative interfaces. See Microsoft documentation for installing Windows Server 2008 SP1, configuring it for dual-stack operation (with a fixed address), and deploying & configuring the DNS server.

If you wish publish the addresses of your external nodes from elsewhere on the Second Internet, this DNS facility can be configured to support internal and external views, and registered as the authoritative server for your domain. You will need to allow incoming connections over DNS (53/udp) through the firewall (over IPv4 and IPv6). I actually recommend that you use Hurricane Electric's free dual-stack DNS service to publish public the external addresses for these labs (as detailed later in this lab).

10.4.1 – DNS Lab 1: Install, Configure for IPv4 Resource Records & Test

Step 1 – Install BIND v9.7 from FreeBSD ports.

Login as *admin* (password *admin$pw*), elevate privilege to root (*su* command, password *root$pw*). Do following commands:

```
# cd /usr/ports/dns/bind97
# make install clean

        (select SSL, REPLACE_BASE and IPV6 options)
...
```

Step 2 – Edit named.conf file (top level BIND configuration file).

```
# cd /var/named/etc/namedb
# edit named.conf          (use whatever editor you prefer)
```

Find the line containing “listen-on” and comment it out

```
old:          listen-on {  127.0.0.1; };

new:          // listen-on { 127.0.0.1; };
```

Find the lines containing “forwarders” remove comment delimiters before and after. Change the dummy IP addresses there to the IP addresses of your ISP’s DNS servers.

```
old:          /*
              forwarders {
                    127.0.0.1;
              };
              */

new:          forwarders {
                    4.2.2.2; 4.2.2.3;
              };
```

Go to the end of the file, and add the following zones:

```
zone “v6lab.com” {
       type master;
       file “/etc/namedb/master/v6lab.com”;
       allow-transfer { localhost; };
       allow-update { key rndc-key; };
};

zone “1.168.192.in-addr.arpa” {
       type master;
       file “/etc/namedb/master/192.168.1.rev”;
       allow-transfer { localhost; };
       allow-update { key rndc-key; };
};
```

Write file, exit editor.

Step 3 – Create rndc (remote name daemon control) configuration on *rndc.key* and concatenate it to *named.conf* file

```
# rndc-confgen -a
# cp named.conf named.conf.old
# cat rndc.key >> named.conf
```

Step 4 – Create forward zone file (v6lab.com)

```
# cd /var/named/etc/namedb/master
# edit v6lab.com          (use any text editor you prefer)

(add the following contents)

$TTL   3600
v6lab.com.          IN     SOA    bsd1.v6lab.com.  root.v6lab.com. (
                                        1            ; Serial
                                        10800        ; Refresh
                                        3600         ; Retry
                                        604800       ; Expire
                                        86400 )      ; Minimum TTL

; nameservers

v6lab.com.          IN     NS     bsd1.v6lab.com.

; Computer names and IP addresses

fw1.v6lab.com.      IN     A      192.168.1.1
bsd1.v6lab.com.     IN     A      192.168.1.11
```

Step 5 – Create reverse zone file (192.168.1.rev)

```
# cd /var/named/etc/namedb/master
# edit 192.168.1.rev             (use any text editor you prefer)

(add the following contents)

$TTL   3600
1.168.192.in-addr.arpa.    IN     SOA    bsd1.v6lab.com.  root.v6lab.com. (
                                        1            ; Serial
                                        10800        ; Refresh
                                        3600         ; Retry
                                        604800       ; Expire
                                        86400 )      ; Minimum TTL

; nameservers

1.168.192.in-addr.arpa.    IN     NS     bsd1.v6lab.com.

; IPv4 addresses

1                          IN     PTR    fw1.v6lab.com.
11                         IN     PTR    bsd1.v6lab.com.
```

Step 6 – Add this FreeBSD server into /etc/resolv.conf (to be your primary DNS server)

```
# edit /etc/resolv.conf

      (add following line as first nameserver)

      nameserver 192.168.1.11
```

Step 7 – Enable BIND (named) in startup script /etc/rc.conf

```
# edit /etc/rc.conf

        (add following line at bottom)

        named_enable=”YES”
```

Step 8 – Check syntax of BIND configuration files without actually starting named:

```
# named-checkconf
```

(fix any problems reported, such as missing periods or semicolons)

Step 9 – Start BIND.

Start the *named* daemon, verify it is running (with *ps*) and test it with nslookup over IPv4. Note – if you have problems, error messages are written to */var/log/messages*.

```
# /etc/rc.d/named start
Starting named.

# ps –ax | grep named
15560  ??  Ss   0:00.06 /usr/sbin/named –t /var/named –u bind

# nslookup – 192.168.1.11

> bsd1.v6lab.com
Server:      192.168.1.11
Address:     192.168.1.11#53

Name:  bsd1.v6lab.com
Address: 192.168.1.11

> fw1.v6lab.com
Server:      192.168.1.11
Address:     192.168.1.11#53

Name:  fw1.v6lab.com
Address: 192.168.1.1

> 192.168.1.11
Server:      192.168.1.11
Address:     192.168.1.11#53

11.1.168.192.in-addr.arpa  name = bsd1.v6lab.com.

> 192.168.1.1
Server:      192.168.1.11
Address:     192.168.1.11#53

1.1.168.192.in-addr.arpa   name = fw1.v6lab.com.

> www.ipv6.org
Server:      192.168.1.11
Address:     192.168.1.11#53
```

```
Non-authoritative answer:
www.ipv6.org  canonical name = shake.stacken.kth.se.
Name:  shake.stacken.kth.se
Address: 130.237.234.40

> exit

#
```

Step 10 – Use netstat to see which sockets BIND is listening to (non-relevant items edited out)

```
# netstat -na
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
tcp4       0      0 127.0.0.1.953          *.*                    LISTEN
tcp4       0      0 127.0.0.1.53           *.*                    LISTEN
tcp4       0      0 192.168.1.11.53        *.*                    LISTEN
udp4       0      0 127.0.0.1.53           *.*
udp4       0      0 192.168.1.11.53        *.*
```

Here is the interpretation of this. You can see that BIND is currently IPv4-only.

IP version	Address	Transport	Port	Service
IPv4	127.0.0.1	tcp	953	rndc
IPv4	127.0.0.1	tcp	53	zone transfers
IPv4	127.0.0.1	udp	53	DNS queries
IPv4	192.168.1.11	tcp	53	zone transfers
IPv4	192.168.1.11	udp	53	DNS queries

10.4.2 – DNS Lab 2: Migrate BIND to Dual Stack (add support for IPv6)

With DNS, the same server that manages IPv4 resource records can also manage IPv6 resource records (unlike DHCP, which required a very different design for DHCPv6). It already accepts queries, and can do zone transfers over either IPv4 or IPv6. All we need to do is add some additional resource records. Note that the reverse zones are done a bit differently in IPv6. For full details, see "BIND and DNS, 5th Edition".

By default, BIND listens for connections from resolvers (DNS clients) only on IPv4. You will also need to make one change in named.conf to make it also listen on IPv6.

Step 1 – Edit *named.conf* file to make it listen on IPv6, then add new reverse zone for IPv6

```
# cd /var/named/etc/namedb
# edit named.conf          (use whatever editor you prefer)
```

Change the "listen-on-v6" line (uncomment it and change "::1" to "any")

```
old:   //    listen-on-v6   { ::1; };

new:   listen-on-v6 { any; };
```

Add the following content after the *1.168.192.in-addr.arpa* zone (modify to reflect your IPv6 prefix, reversed, one hex digit per field):

```
zone “5.6.c.0.5.0.f.1.0.7.4.0.1.0.0.2.ip6.arpa” {
      type master;
      file “/etc/namedb/master/2001.470.1f05.c65.rev”;
      allow-transfer { localhost; };
      allow-update { key rndc-key; };
};
```

Step 2 – Edit master/v6lab.com file to add AAAA records

```
# cd /var/named/etc/namedb/master
# edit v6lab.com
```

Add following new records at the end:

```
fw1.v6lab.com.      IN     AAAA   2001:470:1f05:c65::1
bsd1.v6lab.com.     IN     AAAA   2001:470:1f05:c65::11
```

Step 3 – Create new reverse zone for IPv6 addresses

```
# cd /var/named/etc/namedb/master
# edit 2001.470.1f05.c65.rev     (note use of . instead of : in filename)
```

Enter following information in the file:

```
$TTL   3600
5.6.c.0.5.0.f.1.0.7.4.0.1.0.0.2.ip6.arpa. IN SOA bsd1.v6lab.com. root.v6lab.com. (
                                        1            ; Serial
                                        10800        ; Refresh
                                        3600         ; Retry
                                        604800       ; Expire
                                        86400 )      ; Minimum TTL

; nameservers

5.6.c.0.5.0.f.1.0.7.4.0.1.0.0.2.ip6.arpa.      IN     NS     bsd1.v6lab.com.

; IPv6 addresses

1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0                IN     PTR    fw1.v6lab.com.
1.1.0.0.0.0.0.0.0.0.0.0.0.0.0.0                IN     PTR    bsd1.v6lab.com.
```

Step 4 – Check syntax of BIND configuration files

```
# named-checkconf
```

(fix any problems reported, such as missing periods or semicolons)

Step 5 – Restart the *named* daemon and test it over IPv6

```
# /etc/rc.d/named restart
Starting named.
```

```
# nslookup – 2001:470:1f05:c65::11
> set q=any
> bsd1.v6lab.com
Server:      192.168.1.11
Address:     192.168.1.11#53

bsd1.v6lab.com  has AAAA address 2001:470:1f05:c65::11
Name:  bsd1.v6lab.com
Address: 192.168.1.11

> fw1.v6lab.com
Server:      192.168.1.11
Address:     192.168.1.11#53

bsd1.v6lab.com  has AAAA address 2001:470:1f05:c65::1
Name:  fw1.v6lab.com
Address: 192.168.1.1

> 2001:470:1f05:c65::1
Server:      192.168.1.11
Address:     192.168.1.11#53

1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.5.6.c.0.5.0.f.1.0.7.4.0.1.0.0.2.ip6.arpa
name = fw1.v6lab.com.

> 2001:470:1f05:c65::11
Server:      192.168.1.11
Address:     192.168.1.11#53

1.1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.5.6.c.0.5.0.f.1.0.7.4.0.1.0.0.2.ip6.arpa
name = bsd1.v6lab.com.

> set q=AAAA
> www.ipv6.org
Server:      192.168.1.11
Address:     192.168.1.11#53

Non-authoritative answer:
www.ipv6.org  canonical name = shake.stacken.kth.se.
shake.stacken.kth.se  has AAAA address 2001:6b0:1:eq:202:a5ff:fecd:13a6

Authoritative answers can be found from:

> exit

#
```

Step 6 – Use netstat to see which sockets BIND is listening to (non-relevant items edited out)

```
# netstat -na
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
tcp6       0      0 ::1.953                *.*                    LISTEN
tcp4       0      0 127.0.0.1.953          *.*                    LISTEN
tcp4       0      0 127.0.0.1.53           *.*                    LISTEN
tcp4       0      0 192.168.1.11.53        *.*                    LISTEN
tcp6       0      0 *.53                   *.*                    LISTEN
udp4       0      0 127.0.0.1.53           *.*
udp4       0      0 192.168.1.11.53        *.*
udp6       0      0 *.53                   *.*
```

Here is the interpretation of this (you can see that BIND is now dual-stack):

IP version	Address	Transport	Port	Service
IPv4	127.0.0.1	tcp	953	rndc
IPv4	127.0.0.1	tcp	53	zone transfers
IPv4	127.0.0.1	udp	53	DNS queries
IPv4	192.168.1.11	tcp	53	zone transfers
IPv4	192.168.1.11	udp	53	DNS queries
IPv6	::1	tcp	953	rndc
IPv6	any	tcp	53	zone transfers
IPv6	any	udp	53	DNS queries

10.4.3 – DNS Lab 3: Publish Public IP Addresses on a Dual Stack DNS Service

In addition to providing free tunneled IPv6 service, Hurricane Electric also provides free hosting of dual stack DNS resource records. If you have signed up for their tunneled service, you can also use their free DNS service. You are limited to no more than 25 domains, and only certain resource records are supported (although all you might need for any dual-stack testbed network are included). You need to register the Hurricane Electric nameservers (ns2.he.net and ns3.he.net) as the authoritative DNS servers for your domain (this is done at your domain registrar, e.g. godaddy.com). I recommend registering nameserver addresses with the appropriate TLD DNS servers, which any domain registrar can do. However, only a few can current allow you to register *both* IPv4 *and* IPv6 addresses. One that does is godaddy, so this lab has you obtain a domain from them. If you prefer to use an existing domain, ask your domain registrar if they support registering nameservers with the TLD nameservers with both IPv4 and IPv6 (some do). If not, you can obtain a new domain from any registrar that supports this. If you use any domain registrar other than godaddy, consult that registrar on how to do the steps below.

You will be using the nameservers *ns2.he.net* and *ns3.he.net* at Hurricane Electric to publish your records. These are both dual stack DNS servers (*ns1.he.net* is IPv4 only). As of the writing of this lab, those have the following IP addresses. You should verify that these are still correct when you do the lab, by pinging those nodenames.

nameserver	IPv4 address	IPv6 address
ns2.he.net	216.218.131.2	2001:470:200::2
ns3.he.net	216.218.132.2	2001:470:300::2

Step 1 – Obtain a domain from godaddy (www.godaddy.com).

> Not all TLDs currently support registering of IPv6 nameserver addresses, so choose a domain name that ends in *.com*, *.net* or *.org*, unless you know for certain that your preferred TLD supports this.

Step 2 –Register the external IPv4 addresses of your nameservers with the TLD nameservers.

Bring up the godaddy Domain Manager (*MY PRODUCTS* / *Domains* / *Domain Manager*). Find your new domain name in the list of domains you have obtained from godaddy (e.g. *v6lab.com*) and click on it. In the lower left of the resulting domain specific page, locate the *Host Summary* box. This box contains nodenames for this domain that you have previously registered with the TLD nameservers. With a new domain, there should not be any (it will show *No Hosts*). Click on the *add* link. For *Host name* enter **ns2** (no domain name). For *host IP 1* enter **216.218.131.2**. Do not try to enter the IPv6 address now as *host IP 2* (it won't work). Click on OK. In the confirmation box, click on OK again. Click on the *add* link again (if it is busy, try again in a minute). For Host name, enter **ns3**. For host IP 1, enter **216.218.132.2**. Click on OK. In the confirmation box, click on OK again. Refresh the webpage. You should now see two nodenames in the Host Summary box (NS2 and NS3).

Step 3 – Register the IPv6 addresses of your nameservers with the TLD nameservers.

Click on the *edit* link after NS2. For Host IP 2, enter **2001:470:200::2**. Click on OK. In the confirmation box, click OK again. Click on the *edit* link after NS3. For Host IP 2, enter **2001:470:300::2**. Click on OK. In the confirmation box, click on OK again.

Step 4 – Set authoritative nameservers.

Change the authoritative nameservers for your new domain to the names you just registered (in my case that was *ns2.v6lab.com* and *ns3.v6lab.com*). In the Domain Manager page for your domain name, locate the heading "Nameservers". There will probably be two godaddy nameservers listed there initially (e.g. NS21.DOMAINCONTROL.COM and NS22.DOMAINCONTROL.COM). We will replace those with the HE nameservers, using our registered nameserver names. Click on the *Manage* link under Nameservers. There are four options. Choose "I have specific nameservers for my domain". For Nameserver 1, enter **ns2.v6lab.com** (use your domain name, not v6lab.com), and for Nameserver 2, enter **ns3.v6lab.com** (ditto). Click on OK. In the confirmation box, click on OK again. After a couple of minutes, refresh the webpage. Under Nameservers, you should now see NS2.V6LAB.COM and NS3.V6LAB.COM (only with your domain name instead of v6lab.com). It may take several hours (up to 24) for these records to propagate to the entire Internet (due to DNS caching). Aside from renewing the domain from time to time, or changing the authoritative nameservers for the domain, we will not need to do anything further with this domain at godaddy.

Step 5 – Add your new domain to the free DNS service at Hurricane Electric.

Go to dns.he.net, and log in with the same credentials you used to obtain tunneled service. Click on **Add a new domain.** For *Domain Name* enter the name of your new domain (e.g. v6lab.com), then click on **Add Domain!** You should now see your new domain name in the list of Active Domains for this account. Click on the second icon (edit domain) before that domain name.

Step 6 – Manage the new zone.

First delete ns1.he.net. To do this, click on the "minus sign" in the red circle on that line. You may need to "allow scripted controls" depending on your browser. In the "script prompt" enter the word "DELETE" and click on OK.

Click on the *New A* tab. In the *Create New A Record* window, for *Name* enter **bsd1**. For *Content*, enter the external IPv4 address of your bsd machine (e.g. *12.34.56.101*). Click on the *Submit* button. You should now see the new A record appear.

Click on the *New AAAA* tab. In the *Create New AAAA Record* window, for *Name*, enter **bsd1**. For *Content*, enter the IPv6 address of your bsd server (e.g. *2001:470:1f05:c65::11*). You should now see your new AAAA record appear. Note that there is no "internal" vs "external" IP address in IPv6 (as there is in IPv4) – the same IPv6 address is used internally and externally – NO NAT! Even though you would be using the same IPv6 addresses, you still want a DNS external view (or separate DNS server for external users), which publishes the IPv6 addresses of only those nodes that you want external users to be able to access.

Click on the *New MX* tab. In the *Create New MX Record* window, for *Name*, enter *v6lab.com*. For *Content*, enter *bsd1.v6lab.com*. Leave the other fields as default. Click on the *Submit* button. You should now see your new MX record appear.

Step 7 – Create reverse zone and publish PTR records.

First you need to go back to tunnelbroker.net and bring up your tunnel details. There are five RDNS delegation items. Click on one of them and select *Delegate to HE.NET Nameservers***.** Click on the *Return to Main* link. You should see the five HE nameservers listed there now (ns1.he.net, ..., ns5.he.net). Return to the FreeDNS management webpage. Click on *Add a new reverse*. In the resulting window, for *Prefix*, enter **2001:470:1f05:c65::/64** (use your own routed 64 prefix). Click on the edit icon on that line, and add any IP addresses and the nodenames they should resolve to. For example, for *Address* enter **::11**, and for *Hostname*, enter **bsd1.v6lab.com**. Click on Submit. You should now see information on the added IPv6 PTR record:

Parent Block	Address	Hostname
2001:470:1f05:c65:	:11	bsd1.v6lab.com

Note: providing reverse lookup for your IPv4 addresses will require whoever manages those addresses (probably your IPv4 ISP) to either publish the appropriate PTR records, or delegate the reverse zone for your IP addresses to you to manage. You could delegate them to the HE Free DNS service, if you like.

Step 8 – You can verify this configuration immediately by using nslookup with *ns2.he.net* as the server:

```
C:\>nslookup - ns2.he.net
Default Server:  ns2.he.net
Address:  2001:470:200::2

> bsd1.v6lab.com
Server:  ns2.he.net
Address:  2001:470:200::2

Name:    bsd1.v6lab.com
Addresses:  2001:470:1f05:c65::11
```

```
          120.89.47.101

> 2001:470:1f05:c65::11
Server:  ns2.he.net
Address:  2001:470:200::2

Name:    bsd1.v6lab.com
Address:  2001:470:1f05:c65::11

> set q=mx
> v6lab.com.
Server: ns2.he.net
Address: 2001:470:200::2

v6lab.com    MX preference = 10, mail exchanger = bsd1.v6lab.com
bsd1.v6lab.com      AAAA IPv6 addres = 2001:470:1f05:c65::11
bsd1.v6lab.com      internet address = 12.34.56.101

> exit

C:\>
```

Once these DNS records have propagated, anyone in the world should be able to access your bsd server using your fully qualified domain name (e.g. bsd1.v6lab.com), over IPv4 or IPv6. Again, this is subject to the rules in your firewall.

10.5 – Project 5: Dual Stack Web Server

This project is the simplest way to provide typical dual stack services for both internal and external users. This can be applied to making almost any web application available in dual stack.

The four labs involve deploying Apache 2.2 on FreeBSD. It can easily be deployed on Linux, and information on doing that is widely available online and in other books. The changes described here to support dual-stack operation should work in any deployment of Apache 2.2 (assuming the underlying Operating System supports dual stack).

All popular web browsers support dual-stack operation. I have personally verified the following web browsers to support IPv6:

- Microsoft Internet Explorer 7 and later
- Mozilla Firefox 3.5

This information covers only the basic aspects of installing and running Apache on FreeBSD. Additional documentation on the features used here, and many others, are available online and in various books covering Apache administration.

10.5.1 – Web Server Lab 1: Basic Dual Stack Web Server – Apache on FreeBSD

Step 1 – Create SSL digital certificate for Apache.

There is no passphrase for private key. If you know what you're doing, you can change the Country Name, State, Locality, Org and Org Unit – if you have a real domain, you can change the domain name, but change it everywhere - all other fields should be as shown.

```
# mkdir –p /etc/ssl/apache
# cd /etc/ssl/apache

# openssl genrsa –out serv.key 1024

# openssl req –new –key serv.key –out serv.csr
Country Name (2 letter code) [AU]:PH
State or Province Name (full name) [Some-State]:Cebu
Locality Name (eg, city) []:Cebu City
Organization Name (eg, company) [Internet Widgets Pty Ltd]:InfoWeapons
Organizational Unit Name (eg, section) []:IT
Common Name (eg, YOUR name) []:bsd1.v6lab.com
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:

# openssl x509 –req –days 3650 –in serv.csr –signkey serv.key –out serv.crt
Signtature ok
```

```
subject=/C=PH/ST=Cebu/O=InfoWeapons/OU=IT/CN=www.dummy.org
Getting Private key

# chmod 0400 /etc/ssl/apache/serv.key
# chmod 0400 /etc/ssl/apache/serv.crt
```

Step 2 – Install Apache 2.2 from the FreeBSD ports:

```
# cd /usr/ports/www/apache22
# make all install clean
...

Select: IPv6 (on by default)
```

Step 3 – Edit file */usr/local/etc/apache22/httpd.conf*

```
...
ServerAdmin webmaster@v6lab.com
...
ServerName bsd1.v6lab.com:80
...
Listen 0.0.0.0:80
...
# Various default settings
Include etc/apache22/extra/httpd-default.conf

# Secure (SSL/TLS) connections
Include etc/apache22/extra/httpd-ssl.conf
...
```

Step 4 – Edit file */usr/local/etc/apache22/extra/httpd-ssl.conf*

```
...
Listen 0.0.0.0:443
...
ServerName bsd1.v6lab.com:443
...
ServerAdmin webmaster@v6lab.com
...
SSLCertificateFile “/etc/ssl/apache/serv.crt”
...
SSLCertificateKeyFile “/etc/ssl/apache/serv.key”
...
```

Step 5 – Edit file */etc/rc.conf*, to enable Apache server on boot

```
...
apache22_enable=”YES”
...
```

Step 6 – Check configuration file, fix any reported problems

```
# apachectl configtest
```

Step 7 – Edit file */boot/loader.conf* – to load a couple of kernel modules on boot

```
accf_data_load=”YES”
```

```
accf_http_load="YES"
```

Step 8 – Manually load kernel modules for now (so we don't have to reboot now)

```
# kldload accf_data.ko
# kldload accf_http.ko
```

Step 9 – Start Apache for the first ime, verify httpd is running. If not, check logs in */var/logs*

```
# /usr/local/sbin/apachectl start
# ps –ax | grep httpd
... (should see several instances of httpd running)
```

Step 10 – Test over IPv4.

Use the IPv4 numeric address to insure connection over IPv4. Note that the SSL connection will report an untrusted certificate – allow connection and add an exception).

```
Non-secure: try to surf to http://192.168.1.11
Secure: try to surf to https://192.168.1.11
```

Step 11 – Use netstat to see what ports Apache is listening to (output edited):

```
# netstat -na
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)

tcp4       0      0 *.443                  *.*                    LISTEN
tcp4       0      0 *.80                   *.*                    LISTEN
#
```

Here is the interpretation of this (you can see that Apache is now IPv4-only):

IP version	Address	Transport	Port	Service
IPv4	any	tcp	80	HTTP
IPv4	any	tcp	443	HTTPS

Step 12 – Open appropriate ports in firewall to allow access to web server from outside via IPv4

To do this, surf to the web interface on the m0n0wall (*http://192.168.1.1*), and log in (user = *admin*, password = *mono*).

Create the necessary IPv4 firewall rules: Click on the *Firewall / IPv4 Rules* link. No new LAN rules are required. On the WAN tab, enter the following rules. Open port 80 for HTTP (non-secure) and/or port 443 for HTTPS (secure).

```
Action = Pass, Proto = TCP, Source = any, Port = any,
Destination = 192.168.1.11, Port = 80

Action = Pass, Proto = TCP, Source = any, Port = any,
Destination = 192.168.1.11, Port = 443
```

10.5.2 – Web Server Lab 2: Migrate Apache to Dual Stack

Step 1 – Edit file */usr/local/etc/apache22/httpd.conf* to enable IPv6 for HTTP

```
...
Listen 0.0.0.0:80
Listen [::]:80
...
```

Step 2 – Edit file */usr/local/etc/apache22/extra/httpd-ssl.conf* to enable IPv6 for HTTPS

```
...
Listen 0.0.0.0:443
Listen [::]:443
...
```

Step 3 – Stop and restart Apache, verify httpd is running. If not, check logs in */var/logs*

```
# apachectl stop
# apachectl start
# ps –ax | grep httpd
... (should see several instances of httpd running)
```

Step 4 – Use netstat to see what ports Apache is listening to (output edited):

```
# netstat -na
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)

tcp46      0      0 *.443                  *.*                    LISTEN
tcp46      0      0 *.80                   *.*                    LISTEN
#
```

Here is the interpretation of this (you can see that Apache is now dual-stack):

IP version	Address	Transport	Port	Service
IPv4	any	tcp	80	HTTP
IPv4	any	tcp	443	HTTPS
IPv6	any	tcp	80	HTTP
IPv6	any	tcp	443	HTTPS

Step 5 – Test over IPv6.

Use the numeric IPv6 address to insure use of IPv6. Remember to enclose any numeric IPv6 address in URLs in square brackets. Use the IPv6 address of *your* bsd server. The SSL connection will report an untrusted certificate. Allow the connection and add an exception.

```
Non-secure: try to surf to http://[2001:470:1f05:c65::11]
Secure: try to surf to https://[2001:470:1f05:c65::11]
```

Step 6 – Open appropriate ports in firewall to allow access to web server from outside via IPv6

To do this, surf to the web interface on the m0n0wall (*http://192.168.1.1*), and log in (user = *admin*, password = *mono*).

Create the necessary IPv6 firewall rules: Click on the *Firewall / IPv6 Rules* link. No new LAN rules are required. On the WAN tab, enter the following rules. Open port 80 for HTTP (non-secure) and/or port 443 for HTTPS (secure).

```
Action = Pass, Proto = TCP, Source = any, Port = any,
Destination = 2001:470:1f05:c65::11, Port = 80

Action = Pass, Proto = TCP, Source = any, Port = any,
Destination = 2001:470:1f05:c65::11, Port = 443
```

10.5.3 – Web Server Lab 3: Install PHP, Install PHP test script and run it

Many web applications are built using PHP. This lab shows that it not only runs over IPv6, it can provide information as to the version of IP and the specific address that someone connected from (IPv4 or IPv6). This allows you to generalize web applications to do different things depending on IP version used for the connection (if desired).

Step 1 – Install PHP5 from the FreeBSD ports tree:

```
# cd /usr/ports/lang/php5
# make config ; make install clean

Select: APACHE
```

Step 2 – Create file */usr/local/etc/php.ini*

```
# cd /usr/local/etc
# cp php.ini-recommended php.ini
```

Step 3 – Edit file */usr/local/etc/php.ini* (just created)

```
old:
       ;session.save_path="/tmp"

new:
       session.save_path="/tmp"
```

Step 4 – Edit file */usr/local/etc/apache22/httpd.conf*

```
...
<IfModule dir_module>
       DirectoryIndex index.html index.php index.php5
</IfModule>
...
<IfModule mime_module>
       TypesConfig etc/apache22/mime.types
       AddType application/x-compress .Z
       AddType application/x-gzip .gz .tgz
       AddType application/x-httpd-php .php
```

```
        AddType application/x-httpd-php-source .phps
</IfModule>
```

Step 5 – Reboot the server, then log back in with root privilege

```
# reboot
...
```

Step 6 – Create file */usr/local/www/apache22/data/test.php* with following contents

```
<?php phpinfo(); ?>
```

Step 7 – Test Basic PHP operation on both IPv4 and IPv6

```
Surf to http://192.168.1.11/test.php
Surf to https://192.168.1.11/test.php

Surf to http://[2001:470:1f05:c65::11]/test.php
Surf to https://[2001:470:1f05:c65::11]/test.php
```

Note that the output of phpinfo() includes the IP addresses of the client and server nodes. This will indicate whether you are connecting over IPv4 or IPv6. Look for the variables SERVER_ADDR (the IP address of the Apache server node) and REMOTE _ADDR (the IP address of the client node from which the connection was made). If the connection was over SSL, there will be information about the version of SSL and the certificate(s) used.

10.6 – Project 6: Dual Stack E-mail Server

E-mail is another major service that you can deploy for dual-stack access, for both internal and external users. This is a bit more complicated than providing dual-stack web access, but still not that difficult. You can deploy this with Open Source (Postfix and Dovecot recommended), or using Exchange Server 2007 running on Window Server 2008 SP1. Again, Exchange Server 2007 is available as a free 120 day evaluation. The labs in this book deploy Postfix and Dovecot on FreeBSD. Deploying these on Linux is not much more difficult, but the steps are quite different.

The Open Source Postfix/Dovecot server assumes you have already deployed FreeBSD with dual stack networking. We will also assume that you already have deployed the Apache web server with full support for dual-stack, and have published relevant resource records in a public dual stack DNS (we will add an MX record to those).

It is possible to deploy Postfix and Dovecot in far more sophisticated ways. This deployment has been kept very simple on purpose to allow you to concentrate on the IPv6 aspects, not the E-mail or database aspects of a more complete deployment. In this deployment, the user accounts and passwords are the same as the FreeBSD logins. Mailboxes are in the FreeBSD user's home directory (under directory *mail*). To add another mail user create a new FreeBSD account (using *adduser* or *sysinstall*). To change your mail password, change your FreeBSD password. A more sophisticated deployment might keep usernames and passwords in MySQL, use virtual mailboxes, and even deploy *postfixadmin* for web based administration. There are many online and book sources of information on how to do this. Even the most sophisticated deployments can be migrated to dual stack using the techniques described here.

Most E-mail clients today are dual-stack (they support both IPv4 and IPv6). I have personally verified the following clients to support IPv6 on Windows 7:

- Microsoft Outlook (from Office 2007)
- Microsoft Live Mail Client
- Thunderbird Mail Client

10.6.1 – Mail Server Lab 1: Deploy Postfix MTA for IPv4 Operation

Step 1 – Build and install Postfix from FreeBSD ports (this will also build and install Dovecot)

```
# cd /usr/ports/mail/postfix
# make install clean
       (in postfix options, select DOVECOT and TLS)
       (in dovecot options, accept defaults)
...
You need user "postfix" added to group "mail".
Would you like me to add it [y]? y
Would you like to activate Postfix in /etc/mail/mailer.conf [n]? y
```

Step 2 – Stop sendmail (which is installed by default in FreeBSD)

```
# /etc/rc.d/sendmail forcestop
```

Step 3 – Edit file */etc/rc.conf* to disable sendmail and enable Postfix (add following lines)

```
sendmail_enable="NO"
sendmail_submit_enable="NO"
sendmail_outbound_enable="NO"
sendmail_msp_queue_enable="NO"
postfix_enable="YES"
```

Step 4 – Edit file */etc/periodic.conf*, add following contents

```
daily_clean_hoststat_enable="NO"
daily_status_mail_rejects_enable="NO"
daily_status_include_submit_mailq="NO"
daily_submit_queuerun="NO"
```

Step 5 – Create a self-signed SSL digital certificate (use appropriate country/state/etc)

```
# mkdir -p /etc/ssl/postfix
# cd /etc/ssl/postfix
# openssl req -new -x509 -nodes -out smtpd.pem -keyout smtpd.pem -days 3650
Generating a 1024 bit RSA private key
.....................................................++++++
.................................++++++
writing new private key to 'smtpd.pem'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:PH
State or Province Name (full name) [Some-State]:Cebu
Locality Name (eg, city) []:Cebu City
Organization Name (eg, company) [Internet Widgits Pty Ltd]:InfoWeapons
Organizational Unit Name (eg, section) []:IT
Common Name (eg, YOUR name) []:bsd1.v6lab.com
Email Address []:

# chmod 640 /etc/ssl/postfix/smtpd.pem
# chgrp -R postfix /etc/ssl/postfix
```

Step 6 – Edit file */usr/local/etc/postfix/main.cf*, make appropriate changes shown below

```
softbounce = no

myhostname = bsd1.v6lab.com

mydomain = v6lab.com

myorigin = $myhostname

mydestination = $myhostname, localhost, $mydomain

mynetworks_style = subnet
```

```
mynetworks = 192.168.1.0/24

# TLS CONFIG

smtp_use_tls = yes
smtpd_use_tls = yes
smtp_tls_note_starttls_offer = yes
smtpd_tls_key_file = /etc/ssl/postfix/smtpd.pem
smtpd_tls_cert_file = /etc/ssl/postfix/smtpd.pem
smtpd_tls_CAfile = /etc/ssl/postfix/smtpd.pem
smtpd_tls_loglevel = 0
smtpd_tls_received_header = yes
smtpd_tls_session_cache_timeout = 3600s
tls_random_source = dev:/dev/urandom
```

Step 7 – Edit file */usr/local/etc/postfix/master.cf*

```
...
smtps     inet  n       -       n       -       -       smtpd
  -o smtpd_tls_wrappermode=yes
...
```

Step 8 – Edit file */etc/aliases*

Change "root" to an email address you want system messages to be mailed to:

```
root: webmaster@v6lab.com
```

Step 9 – Create aliases.db file from /etc/aliases file

```
# /usr/bin/newaliases
```

Step 10 – Start postfix manually:

```
# /usr/local/etc/rc.d/postfix start
```

Step 11 – Test postfix MTA by doing telnet to port 25 using IPv4 address

```
# telnet 192.168.1.11 25
Trying 192.168.1.11...
Connected to bsd1.v6lab.com.
Escape character is '^]'.
220 bsd1.v6lab.com ESMTP Postfix
ehlo x.com
250-bsd1.v6lab.com
250-PIPELINING
250-SIZE 10240000
250-VRFY
250-ETRN
250-STARTTLS
250-ENHANCEDSTATUSCODES
250-8BITMIME
250 DSN
quit
221 2.0.0 Bye
Connection closed by foreign host.
```

```
#
```

10.6.2 – Mail Server Lab 2: Deploy Dovecot POP3/IMAP Mail Retrieval Server

Dovecot is a very powerful server that implements both the POP3 and IMAP mail retrieval from mailboxes created and loaded by the Postfix MTA. It supports IPv4 and IPv6 (both with and without TLS). Note that Dovecot was built and installed in the previous lab, in addition to Postfix. Now we will configure and start it.

Step 1 – Edit file */etc/rc.conf* to start Dovecot at boot

```
...
dovecot_enable="YES"
...
```

Step 2 – Copy Dovecot configuration files

```
# cd /usr/local/etc
# cp /usr/local/share/examples/dovecot/dovecot.conf /etc/local/etc
```

Step 3 – Edit file */usr/local/etc/dovecot.conf*, make the following changes:

```
protocols = imap imaps pop3 pop3s

ssl = yes
ssl_cert_file = /etc/ssl/postfix/smptd.pem
ssl_key_file = /etc/ssl/postfix/smtpd.pem

login_greeting = Dovecot Ready.

protocol lda {
  postmaster_address = postmaster@v6lab.com
}

auth default {
  mechanisms = plain login

  client {
      path = /var/spool/postfix/private/auth
      mode = 0660
      user = postfix
      group = postfix
  }
}
```

Step 4 – Start Dovecot

```
# /usr/local/etc/rc.d/dovecot start
```

If there are problems, look in /var/log/maillog

Step 5 – Use netstat to see what ports Postfix and Dovecot are listening to (edited):

```
# netstat -na
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
tcp4       0      0 *.995                  *.*                    LISTEN
tcp4       0      0 *.110                  *.*                    LISTEN
tcp4       0      0 *.993                  *.*                    LISTEN
tcp4       0      0 *.143                  *.*                    LISTEN
tcp4       0      0 *.465                  *.*                    LISTEN
tcp4       0      0 *.25                   *.*                    LISTEN
```

Here is the interpretation of this (you can see that email is now IPv4-only):

IP version	Address	Transport	Port	Service
IPv4	any	tcp	25	SMTP, SMTPS
IPv4	any	tcp	465	alternative SMTP
IPv4	any	tcp	110	POP3
IPv4	any	tcp	995	POP3S
IPv4	any	tcp	143	IMAP
IPv4	any	tcp	993	IMAPS

Step 6 – Test POP3 protocol by connecting with telnet on port 110, using IPv4

```
# telnet 192.168.1.11 110
Trying 192.168.1.11...
Connected to bsd1.v6lab.com.
Escape character is '^]'.
+OK Dovecot ready.
user admin
+OK
pass admin$pw
+OK Logged in.
list
+OK 0 messages:
.
quit
+OK Logging out.
Connection closed by foreign host.
#
```

Step 7 – Test IMAP protocol by connecting with telnet on port 143, using IPv4

```
# telnet 192.168.1.11 143
Trying 192.168.1.11...
Connected to bsd1.v6lab.com.
Escape character is '^]'.
* OK [CAPABILITY IMAP4rev1 LITERAL+ SASL-IR LOGIN-REFERRALS ID ENABLE STARTTLS
AUTH=PLAIN AUTH=LOGIN] Dovecot ready.
100 login admin admin$pw
100 OK [CAPABILITY IMAP4rev1 LITERAL+ SASL-IR LOGIN-REFERRALS ID ENABLE SORT
SORT=DISPLAY THREAD=REFERENCES THREAD=REFS MULTIAPPEND UNSELECT IDLE CHILDREN
NAMESPACE UIDPLUS LIST-EXTENDED I18NLEVEL=1 CONDSTORE QRESYNC ESEARCH ESORT
SEARCHRES WITHIN CONTEXT=SEARCH LIST-STATUS] Logged in
200 logout
* BYE Logging out
200 OK Logout completed.
Connection closed by foreign host.
```

```
#
```

Step 8 – Test mail server with email client using POP3 and IMAP, using IPv4

Configure any email client (e.g. Outlook) as follows:

```
Account type                    POP3
Incoming mail server            192.168.1.11
Outgoing mail server (SMTP)     192.168.1.11
User Name                       System Administrator
Login                           admin
Password                        admin$pw
```

Try sending and retrieving messages from your Windows node inside the LAN.

Change account type to IMAP and repeat test

Step 9 – Open appropriate ports in firewall to allow access to mail server from outside

To do this, surf to the web interface on the m0n0wall (*http://192.168.1.1*), and log in (user = *admin*, password = *mono*).

Create the necessary IPv4 firewall rules: Click on the *Firewall / IPv4 Rules* link. No new LAN rules are required. On the WAN tab, enter the following rules. You need to allow at least incoming SMTP. You can allow incoming POP3 and/or IMAP if you want people to be able to retrieve their email from outside the LAN, in either plaintext (110, 143) and/or secure versions (995, 993). Some administrators allow POP3S and IMAPS from outside, but POP3 and IMAP only from the LAN. In that case, you would allow incoming SMTP, POP3S and IMAPS, but not POP3 or IMAP.

```
Action = Pass, Proto = TCP, Source = any, Port = any,
Destination = 192.168.1.11, Port = 25

Action = Pass, Proto = TCP, Source = any, Port = any,
Destination = 192.168.1.11, Port = 110

Action = Pass, Proto = TCP, Source = any, Port = any,
Destination = 192.168.1.11, Port = 143

Action = Pass, Proto = TCP, Source = any, Port = any,
Destination = 192.168.1.11, Port = 995

Action = Pass, Proto = TCP, Source = any, Port = any,
Destination = 192.168.1.11, Port = 993
```

Step 10 – Publish MX record in public DNS server

This allows other mail serves to determine the preferred mail server(s) for your domain.

When a local user (one with an account on this email server) sends a message to another local user, the email never leaves the LAN. It is sent from the first user to the mail server via SMTP, IMAP or webmail,

the immediately delivered to the mailbox of the local recipient, who can retrieve it with POP3, IMAP or possibly webmail.

When a remote user (one with an account on some other email server) sends a message to one of your local users, they deliver the message to *their* email server via SMTP, IMAP or webmail. Sometime later, their mail server will transfer outgoing mail to your domain by doing a DNS retrieval to find the MX record for your domain name. It will get a prioritized list of names of preferred mail servers for your domain (often only one). For each mailserver name (until it succeeds, or runs out of names) it will then retrieve A/AAAA records, and obtain a list of IPv4 and/or IPv6 addresses for that name. If IPv6 transport is available, it will try IPv6 first then fail back to IPv4. If no IPv6 transport is available, it will try IPv4 immediately.

10.6.3 – Mail Server Lab 3: Migrate Postfix and Dovecot to Dual Stack

Now that we have a working IPv4-only basic E-mail server up and running, let's migrate it to dual stack!

Step 1 – Edit file /usr/local/etc/postfix/main.cf

After "inet_interfaces" section, add the following lines:

```
# inet_protocols – specify which IP address families to support
# options: "IPv4", "IPv6", "IPv4, IPv6", "all"
inet_protocols = all
```

Add your IPv6 networks to *mynetworks* specification:

```
mynetworks = 127.0.0.0/8, 192.168.1.0/24,
      [IPv6:::1/128], [IPv6:2001:470:1f05:c65::/64]
```

Step 2 – Edit file /usr/local/etc/dovecot.conf

Add IPv6 unspecified address to Listen list (be sure to uncomment line):

```
listen = *, [::]
```

Step 3 – Retart Postfix and Dovecot servers

```
# /usr/local/etc/rc.d/postfix restart
postfix/postfix-script: stopping the Postfix mail system
postfix/postfix-script: starting the Postfix mail system

# /usr/local/etc/rc.d/dovecot restart
Stopping dovecot.
Starting dovecot.
```

Step 4 – Use netstat to verify which ports Postfix and Dovecot are listening to (edited)

```
# netstat -na
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
```

```
tcp6       0      0 *.995                  *.*                    LISTEN
tcp4       0      0 *.995                  *.*                    LISTEN
tcp6       0      0 *.110                  *.*                    LISTEN
tcp4       0      0 *.110                  *.*                    LISTEN
tcp6       0      0 *.993                  *.*                    LISTEN
tcp4       0      0 *.993                  *.*                    LISTEN
tcp6       0      0 *.143                  *.*                    LISTEN
tcp4       0      0 *.143                  *.*                    LISTEN
tcp6       0      0 *.465                  *.*                    LISTEN
tcp4       0      0 *.465                  *.*                    LISTEN
tcp6       0      0 *.25                   *.*                    LISTEN
tcp4       0      0 *.25                   *.*                    LISTEN
```

Here is the interpretation of this (you can see that email is now dual-stack):

IP version	Address	Transport	Port	Service
IPv4	any	tcp	25	SMTP, SMTPS
IPv4	any	tcp	465	alternative SMTP
IPv4	any	tcp	110	POP3
IPv4	any	tcp	995	POP3S
IPv4	any	tcp	143	IMAP
IPv4	any	tcp	993	IMAPS
IPv6	any	tcp	25	SMTP, SMTPS
IPv6	any	tcp	465	alternative SMTP
IPv6	any	tcp	110	POP3
IPv6	any	tcp	995	POP3S
IPv6	any	tcp	143	IMAP
IPv6	any	tcp	993	IMAPS

Step 5 – Test SMTP protocol by connecting with telnet on port 25 using IPv6 address

```
# telnet 2001:470:1f05:c65::11 25
Trying 2001:470:1f05:c65::11...
Connected to 2001:470:1f05:c65::11.
Escape character is '^]'.
220 bsd1.v6lab.com ESMTP Postfix
ehlo x.com
250-bsd1.v6lab.com
250-PIPELINING
250-SIZE 10240000
250-VRFY
250-ETRN
250-STARTTLS
250-ENHANCEDSTATUSCODES
250-8BITMIME
250 DSN
quit
221 2.0.0 Bye
Connection closed by foreign host.
#
```

Step 6 – Test POP3 protocol by connecting with telnet on port 110 using IPv6 address

```
# telnet 2001:470:1f05:c65::11 110
Trying 2001:470:1f05:c65::11...
Connected to 2001:470:1f05:c65::11.
Escape character is '^]'.
+OK Dovecot ready.
user admin
+OK
pass admin$pw
+OK Logged in.
list
+OK 0 messages:
.
quit
+OK Logging out.
Connection closed by foreign host.
#
```

Step 7 – Test IMAP protocol by connecting with telnet on port 143 using IPv6 address

```
# telnet 2001:470:1f05:c65::11 143
Trying 2001:470:1f05:c65::11...
Connected to 2001:470:1f05:c65::11.
Escape character is '^]'.
* OK [CAPABILITY IMAP4rev1 LITERAL+ SASL-IR LOGIN-REFERRALS ID ENABLE STARTTLS
AUTH=PLAIN AUTH=LOGIN] Dovecot ready.
100 login admin admin$pw
100 OK [CAPABILITY IMAP4rev1 LITERAL+ SASL-IR LOGIN-REFERRALS ID ENABLE SORT
SORT=DISPLAY THREAD=REFERENCES THREAD=REFS MULTIAPPEND UNSELECT IDLE CHILDREN
NAMESPACE UIDPLUS LIST-EXTENDED I18NLEVEL=1 CONDSTORE QRESYNC ESEARCH ESORT
SEARCHRES WITHIN CONTEXT=SEARCH LIST-STATUS] Logged in
200 logout
* BYE Logging out
200 OK Logout completed.
Connection closed by foreign host.
#
```

Step 8 – Open appropriate ports in firewall to allow access to mail server from outside

To do this, surf to the web interface on the m0n0wall (*http://192.168.1.1*), and log in (user = *admin*, password = *mono*).

Create the necessary IPv6 firewall rules: Click on the *Firewall / IPv6 Rules* link. No new LAN rules are required. On the WAN tab, enter the following rules. You need to allow at least incoming SMTP. You can allow incoming POP3 and/or IMAP if you want people to be able to retrieve their email from outside the LAN, in either plaintext (110, 143) and/or secure versions (995, 993). Some administrators allow POP3S and IMAPS from outside, but POP3 and IMAP only from the LAN. In that case, you would allow incoming SMTP, POP3S and IMAPS, but not POP3 or IMAP.

Action = *Pass*, Proto = *TCP*, Source = *any*, Port = *any*,
Destination = *2001:470:1f05:c65::11*, Port = *25*

Action = *Pass*, Proto = *TCP*, Source = *any*, Port = *any*,
Destination = *2001:470:1f05:c65::11*, Port = *110*

Action = *Pass*, Proto = *TCP*, Source = *any*, Port = *any*,
Destination = *2001:470:1f05:c65::11*, Port = *143*

```
Action = Pass, Proto = TCP, Source = any, Port = any,
Destination = 2001:470:1f05:c65::11, Port = 995

Action = Pass, Proto = TCP, Source = any, Port = any,
Destination = 2001:470:1f05:c65::11, Port = 993
```

10.6.4 – Mail Server Lab 4: Deploy Squirrelmail Webmail Access

Squirrelmail is a web based application written in PHP5. It is typical of many web based applications. It has no specific support for IPv6, but because you are installing it on a web server (Apache) that has been configured for dual-stack, it works dual stack with no further effort (unless they display or allow you to specify numeric IP addresses). Most web applications "get a free ride" in this manner. The same is true for Microsoft IIS based web applications. Once IIS is configured to run dual-stack, most IIS web applications will run dual-stack with no modifications.

Step 1 – Build and install SquirrelMail from the FreeBSD ports.

```
# cd /usr/ports/mail/squirrelmail
# make config ; make install clean
```

Step 2 – Increase the upload file size limit from 2MB to 8MB.

```
# edit /usr/local/etc/php.ini

old:   upload_max_filesize = 2M

new:   upload_max_filesize = 8M
```

Step 3 – Run the menu based SquirrelMail configuration utility.

```
# cd /usr/local/www/squirrelmail
# ./configure

      Select menu item 2 (Server Settings)
      Select menu item 1 (Domain)
      Specify domain name as v6lab.com

      Select menu item 2 (Server Settings)
      Select menu item A (Update IMAP Settings)
      Select menu item 8 (Server Software)
      Specify server software as dovecot

      Select menu item Q (Quit)
```

Step 4 – Create Apache configuration file for SquirrelMail.

Apache will load any configuration files found in directory /usr/local/etc/apache22/Includes. Create the file squirrelmail.conf.

```
# edit /usr/local/etc/apache22/Includes/squirrelmail.conf
```

Add the following content:

```
Alias /squirrelmail "/usr/local/www/squirrelmail/"

<Directory "/usr/local/www/squirrelmail/">
Options None
AllowOverride None
Order allow,deny
Allow from all
</Directory>
```

Step 5 – Restart Apache

```
# /usr/local/etc/rc.d/apache22 restart
Performing sanity check on apache22 configuration:
Syntax OK
Stopping apache22.
Waiting for PIDS: 956.
Performing sanity check on apache22 configuration:
Syntax OK
Starting apache22.
#
```

Step 6 – Test over IPv4 and IPv6 (https will also work):

```
http://192.168.1.11/squirrelmail

http://[2001:470:1f05:c65::11]/squirrelmail
```

Once you are connected to SquirrelMail, login in with username *admin*, password *admin$pw*. Try sending and reading some emails. These will need to be sent to yourself (admin@v6lab.com), or you can create some more E-mail users by creating more FreeBSD logins (try *sysinstall* for menu based user management). If you have published your external A, AAAA and MX records on a public DNS server (as covered), and you have opened the appropriate ports on your firewall, your mail server can actually exchange mail with the outside world. Some mail servers may reject outgoing mail from your server if you don't manage to publish the reverse DNS records for your public IPv4 addresses (this is not covered in the labs, as the records in question are managed by your IPv4 ISP).

If you need access to another IPv6 capable mail server to exchange mail with, feel free to send me mail at lhughes@infoweapons.com or lhughes@hughesnet.org. Both are fully dual stacked. I will reply to all messages sent over IPv6.

Step 7 – Review headers of a message sent over IPv6

Note: this message was sent from a dual-stack Postfix/Dovecot/Squirrelmail mail server to Exchange 2007 running on Windows Server 2008 SP1. This Postfix server includes SASL authentication, otherwise is exactly like the one installed here. Here are the complete E-mail headers.

```
MIME-Version: 1.0
Received: from us1.homev6.com (2001:418:5403:3000::d) by ws2.infoweapons.com
```

```
 (2001:418:5403:2400::10:11) with Microsoft SMTP Server (TLS) id 8.2.254.0;
 Tue, 15 Jun 2010 10:53:18 +0800
Received: from us1.homev6.com (localhost [IPv6:::1]) (Authenticated sender:
 lhughes@homev6.com)       by us1.homev6.com (Postfix) with ESMTPA id 4D6A7228C0;
       Wed,  9 Jun 2010 09:56:32 +0800 (PHT)
Received: from 2001:418:5403:3000:64b6:40f9:5a20:8e5        (SquirrelMail
 authenticated user lhughes@homev6.com)        by us1.homev6.com with HTTP;
        Wed, 9 Jun 2010 09:56:32 +0800
From: "lhughes@homev6.com" <lhughes@homev6.com>
To: Lawrence Hughes <lhughes@infoweapons.com>
CC: "lhughes@hughesnet.org" <lhughes@hughesnet.org>
Date: Wed, 9 Jun 2010 09:56:32 +0800
Subject: test from homev6.com squirrelmail
Thread-Topic: test from homev6.com squirrelmail
Thread-Index: AcsMNek444X4LFI5ShmQMwQtkcmv9w==
Message-ID: <47abce3c4bc96bf3fc494417470c8532.squirrel@us1.homev6.com>
Accept-Language: en-US
Content-Language: en-US
X-MS-Exchange-Organization-AuthAs: Anonymous
X-MS-Exchange-Organization-AuthSource: ws2.infoweapons.com
X-MS-Has-Attach:
X-MS-TNEF-Correlator:
user-agent: SquirrelMail/1.4.20-RC2
Content-Type: text/plain; charset="iso-8859-1"
Content-Transfer-Encoding: quoted-printable
```

Received headers are in reverse order (the first one is the most recent). So, starting with the last Received header and working backwards:

- Message was received from a node with IPv6 address *2001:418:5403:3000:64b6:40f9:5a20:8e5* (which happens to be my Windows 7 workstation) by Squirrelmail running on us1.homev6.com (over HTTP, because this was submitted via Squirrelmail, as shown by "user-agent:"). This link used SASL authentication (*Authenticated user lhughes@homev6.com*).
- Message was then received from Squirrelmail running on us1.homev6.com with IPv6 address *::1* (localhost) by Postfix running on the same node. Again, SASL authentication was used (*Authenticated sender: lhughes@homev6.com*). Sometime later, Postfix relayed the message onward since it was not addressed to a local domain. To do this, it did a DNS query for MX records for the destination domain (infoweapons.com) and found the preferred mail server for that domain. This nodename has both IPv4 (A) and IPv6 (AAAA) records defined for it, so it chose the preferred IPv6 address of the InfoWeapons Exchange server.
- Message was then received from us1.homev6.com with IPv6 address *2001:418:5403:3000::d* (still in my home LAN) by Exchange Server (Microsoft SMTP server) over TLS, at IPv6 address *2001:418:5403:2400::10:11* (in the InfoWeapons LAN).
- I then retrieved the message from the Exchange server at InfoWeapons (which does not add another Received header). Even though it is not documented in the headers, that retrieval was also done over IPv6 (using IMAP). Therefore this message went over IPv6 all the way from sender to recipient.

10.7 – Conclusion

If you have done all of the above labs, you now have a fairly complete dual-stack testbed network, and are familiar with many of the things that you will need to do as a network administrator. Between the labs and the book, hopefully you now understand the following things:

- It is not particularly difficult to obtain free tunneled IPv6 service, even using free components. You do not need to wait for your ISP to provide IPv6 service to go fully operational. Simple transition mechanisms simplify the migration to full dual stack operation.
- Most operating systems, and many existing network applications (BIND, Apache, Postfix, Dovecot, ssh/sshd, etc.) are already fully capable of supporting full dual-stack operation. Network configuration is not that different from IPv4.
- Most web applications (Apache or IIS based) get a "free ride", once the underlying web server has been migrated to dual-stack. In addition (although not covered in these labs), most Microsoft ".Net" applications get a free ride.
- IPv4 NAT really doesn't provide *any* useful function other than extending the life of the IPv4 address space, and only then at very high price (in terms of lost capabilities and additional complexity). It adds *no* security in firewall architectures. NAT is a crutch you no longer need. IPv6 without NAT actually provides a simpler, better firewall architecture (no need for BINAT, proxy ARP, NAT traversal, etc.). We are really just returning to the pre-NAT "classical" firewall architectures, not something new and untested.
- There are only a few really new concepts in IPv6 that current network administrators need to master, such as tunneling, Application Layer Gateways, hexadecimal address representation, address scopes (e.g. link-local addresses), working without NAT, needing to provide Router Advertisement messages (for SLAAC to work), multicast and IPsec that actually work, etc. Everything else is remarkably similar to working with IPv4.
- The supply of IPv4 addresses is *really* almost gone, and there is no alternative to this other than migration to IPv6. The timeline on this is sooner than most people realize. Probably before the end of 2011, the last address will have been allocated, and you have better be ready to support IPv6 if you want to keep your job (or have your organization continue operation) past that point.

Congratulations, and welcome to the Second Internet as its newest citizen!

Appendix A – Cryptography & PKI

Several topics in this book involve cryptography and/or PKI (Public Key Infrastructure). IPsec (Internet Protocol layer security) includes both AH (Authentication Header) and ESP (Encapsulating Security Payload). AH is used to do sender authentication of packets, and detection of tampering. Instead of using a combination of message digest (e.g. SHA1) with asymmetric cryptography (e.g. RSA), AH uses a *keyed message digest*, also called Hashed Message Authentication Code (HMAC). ESP does symmetric key encryption of the payload part of an IP packet. The symmetric key for this can be manually distributed (referred to as "shared secret key"), or automatically exchanged in a secure manner using Internet Key Exchange (IKE). IKE uses an application of asymmetric key cryptography called the Diffie-Hellman Key Agreement algorithm, and a new kind of digital certificate called an IPsec Certificate to fully automate distribution of symmetric keys for IPsec.

The cryptographic mechanisms used in these IP security protocols are explained in this chapter. If you are not already familiar with them, you should review this chapter before trying to understand the IP security protocols themselves. There are some good books listed in the bibliography that cover these cryptographic mechanisms in considerably more detail, if this chapter is not sufficient for you. I spent two years with VeriSign creating and delivering training on just these topics, and took an entire week to cover the topics in this chapter. What follows is a very concise coverage of these topics.

A.1 – Cryptography Standards

The X.509 digital certificate is the foundation of PKI. It was originally specified as part of the ITU-T (formerly CCITT) X.500 standard for directory services. The public key certificate was specified in part of this larger standard, as fascicle X.509, "The Directory: Public-key and attribute certificate frameworks", July 1998. What is in current use is version 3 of this specification. This standard has been incorporated into the RFC standard set as RFC 5280, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", May 2008. The standards related to PKI and cryptography based on X.509 certificates are overseen by the PKIX Working Group of the IETF.

Since X.500 was part of the OSI network standard, there are some artifacts of that origin in the X.509 certificate, such as *distinguished names*, for example:

> cn=Lawrence Hughes, o=InfoWeapons Corp., ou=Administration, c=PH

RFC 5280 describes ways to integrate this OSI technology into TCP/IP, in addition to making all of the details of the certificate itself available to all for free. If this syntax looks familiar, it is also found in LDAP, which also was derived from the OSI X.500 technology.

Some of the concepts covered in this chapter were first defined in a set of standards from RSA called PKCS (Public Key Cryptography Standards). For example, S/MIME was first defined in PKCS #7. There were a number of PKCS standards, the most important of which were:

- PKCS #1 – RSA Cryptography Standard, now in **RFC 3447, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1", February 2003**.
- PKCS #3 – Diffie-Hellman Key Agreement Standard
- PKCS #5 – Password-based Encryption Standard, now in RFC 2898, "PKCS #5: Password-Based Cryptography Specification Version 2.0", September 2000
- PKCS #7 – Cryptographic Message Syntax Standard, now in RFC 2315, "PKCS #7: Cryptographic message Syntax Standard Version 1.5", March 1998. The specifics of S/MIME, which is a large part of this standard have been refined in RFC 3851, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification", July 2004; and RFC 3852, "Cryptographic Message Syntax (CMS)", July 2008. The most recent specification of S/MIME is RFC 5751, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification", January 2010.
- PKCS #8 – Private-Key Information Syntax Standard, now in RFC 5208, "Public-Key Cryptography Standards (PKCS) #8: Private-Key Information Syntax Specification Version 1.2", May 2008
- PKCS #10 – Certification Request Standard, now in **RFC 2986, "PKCS #10: Certification Request Syntax Specification Version 1.7", November 2000**
- PKCS #11 – Cryptographic Token Interface (Cryptoki)
- PKCS #12 – Personal Information Exchange Syntax Standard
- PKCS #15 – Cryptographic Token Information Format Standard, now ISO/IEC 7816-15.

The original PKCS standards are still available today if you are interested, but most of the important ones have been incorporated into the RFC standards track and continue to evolve there. However, even today, we often refer to creating a PKCS #10 *Certificate Signing Request*, using the PKCS #11 *Cryptoki Application Program Interface* to a cryptographic module, or exporting public and/or private keys securely in a PKCS #12 package.

A.2 – Cryptography, Encryption and Decryption

Cryptography involves scrambling information in such a way that all of the information is still present, but recoverable only by a recipient who has access to the scheme by which it can be unscrambled. In a very simple case, alphabetic substitution could map each letter of the alphabet to the letter 3 further along, so that A becomes D, B becomes E, etc. That is called an *encryption algorithm*. Here is the complete mapping from the letters in the message (the *plaintext*) to the encrypted letters (the *ciphertext*) for this scheme. To encrypt a message, you would take each letter (in the first line) and map it to the one below it (in the second line).

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
D E F G H I J K L M N O P Q R S T U V W X Y Z A B C
```

The word "HELLO" would become "KHOOR". To unscramble this message, the recipient would need to know how to reverse the encryption (a *decryption algorithm*), which in this case would involve mapping each letter to the one three *before* each of the encrypted letters. So, D would become A, E would become B, etc. You could use the same two lines, but instead locate each letter from the ciphertext in the *second* line and replace it with the letter *above* it (in the *first* line). Hence "KHOOR" would become "HELLO".

In this case, the algorithm must be kept secret, because knowledge of it would allow anyone to decrypt the message. The security of this system depends on keeping the *algorithm* secret.

A.2.1 – Cryptographic Keys

The "shift by 3" encryption system is so simple that almost anyone could figure out how to *crack* a message encrypted with it in just a few minutes. To make it a stronger system, instead of always shifting by *three* characters, you could shift by *n* letters (where n could be any number from 0 to 25). The "shift by 3" scheme would be a special case of this "shift by N" scheme, where N = 3. In the "shift by N" scheme, the value of N is known as an *encryption key*. To decrypt a message, the recipient needs to know not only the decryption algorithm, but also the specific *key* used to encrypt a given message. One day, the sender might use N=7 ("HELLO" becomes "OLSSV"), but on the next, they might use N=12 ("HELLO" becomes "TQXXA"). The same plaintext results in different ciphertexts, depending on the key used.

In general, real encryption algorithms are *key driven*. The same algorithm can produce radically different ciphertexts, depending on which key is used. A key is really just an ordered string of bits. You might think of the encryption algorithm as a generalized machine, in which case a particular key would be like one of many possible cams, which control the operation of the machine (how it slices and dices the data). Replace the cam with a different one and the same machine slices and dices the data in a completely different way.

In fact, modern algorithms are designed in such a way that complete knowledge of the general algorithm itself can be made available to anyone, and it does not make it any easier for them to crack a message encrypted with a key that they do not know. In other words, the security of the system is entirely dependent on keeping the *keys* used (not the details of the algorithm) secret. Full details of most modern encryption algorithms are not only known, they are standardized, peer reviewed, and available to anyone that is interested. The more good cryptanalysts that have reviewed an algorithm, and pronounced it strong (difficult to crack), the better. It is not possible to prove than an algorithm is uncrackable, but you can build evidence that as yet, no one has managed to crack it (or at least, no one has *admitted* to being able to crack it).

A.2.2 – Symmetric Key Cryptography

In the "shift by N" system, the same key that was used to encrypt some plaintext is also used to decrypt the resulting ciphertext. This is analogous to your house key. You use the same key to lock your house when you leave, and to unlock it when you return. This is the defining characteristic of *symmetric key cryptography* which has been around for about 2,000 years. In comparison, with asymmetric key cryptography (invented only recently), one key of a related pair of keys is used to encrypt some plaintext, but *only the other key the pair* can decrypt the resulting ciphertext. Not even the key it was encrypted with can decrypt it. This is not as intuitive as symmetric key cryptography, but allows some remarkable things to be done.

Standards related to symmetric key cryptography include:

- FIPS PUB 46, Data Encryption Standard (January 1977)
- FIPS PUB 46-1, Data Encryption Standard (January 1988)
- FIPS PUB 46-2, Data Encryption Standard (December 1993)
- **FIPS PUB 46-3, Data Encryption Standard (October 1999) (includes Triple DES)**
- **FIPS PUB 197, Advanced Encryption Standard (November 2001)**
- RFC 3565, "Use of the Advanced Encryption Standard (AES) Encryption Algorithm in Cryptographic Message Syntax", July 2003
- **RFC 3566, "The AES-XCBC-MAC-96 Algorithm and Its Use With IPsec", September 2003**
- **RFC 3602, "The AES-CBC Cipher Algorithm and Its Use with IPsec", September 2003**

A.2.3 – Cryptanalysis

The algorithm just described ("shift by N") could be known by the recipient in advance, but somehow the sender would have to let the recipient know which key (which value of N) was used to encrypt a given message. There are only 26 possible keys (0 to 25), which would only take 5 bits to represent (actually 5 bits would allow up to 32 possible keys). A *key length* of 5 bits is not very secure. It would not take long to try every possible key, and chances are, only one of the possible values would result in an English text. If the message was longer, or the same key was used for all messages on a given day, once a third party figured out the key being used, they could easily decrypt the rest of the message, or all of the messages encrypted with it.

Note that in the "shift by N" system, the key N=0 is a *weak* key, as the ciphertext is identical to the plaintext. Real encryption algorithms may have weak keys as well. DES has 16 known weak keys, which you have to check for when you generate a random DES key (although the probability of creating one is quite low). It is amazing that after all the slicing and dicing that DES does on the binary level, the result (using a weak key) could be exactly what you started with, but that is the way it works.

Cracking secure messages (being able to recover the plaintext of an encrypted message without having access to the correct key) is called *cryptanalysis*. There are many techniques used to cryptanalyse encrypted messages. For example, with alphabetic substitution schemes, letter frequencies can help you identify the encrypted characters for the most common letters (the character that appears most often in the ciphertext is probably the code for E, etc.). With modern algorithms there are other techniques. With DES, a technique called differential cryptography was used to determine its real strength only justified a 56-bit key – anything longer would only give the illusion of more strength. Sometimes cryptanalysts can completely cryptanalyse a new proposed algorithm, which means it is cracked, and messages can be decrypted without knowledge of the keys used. You might think of cryptography as building secure safes and cryptanalysis as safe cracking. Cryptology is the science that includes both cryptography and cryptanalysis. RSA (the company) was founded by two cryptographers and one cryptanalyst. The cryptographers proposed one asymmetric key system after another, then the cryptanalyst would analyze and break them. The cryptographers would then try to design one more difficult to break, based on how the previous system was broken. When an algorithm remained unbroken after quite an effort, they released it as the RSA algorithm. Since then, many other cryptanalysts have tried to break it, with no success. The RSA algorithm is now specified in RFC 4055,

"Additional Algorithms and Identifiers for RSA Cryptography for use in the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", June 2005.

The "shift by N" encryption scheme can be implemented with two disks or two rings, each of which has the entire alphabet written on them, but one of which can be rotated to put A on the plaintext ring next to any letter on the ciphertext ring. Various Radio and TV shows (as well as Cereal manufacturers) have included these *cipherdisks* as premiums over the years. It is a real (but not very strong) cryptographic system, complete with the concept of a key.

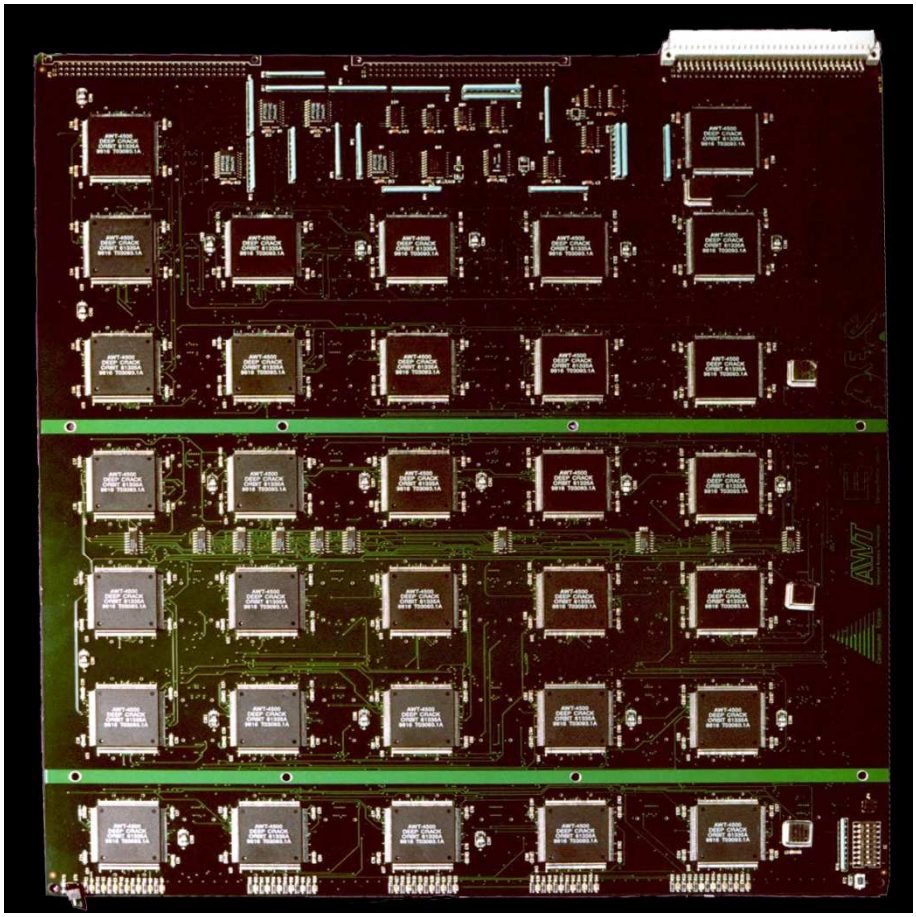
A Secret Decoder Ring
From Wikipedia Creative Commons

A.2.4 – Cryptographic Strength

Modern real world cryptographic systems don't do just alphabetic substitution, or work on just ASCII text. They accept a block of binary data as input, and scramble information in very complex ways at the bit level. The output is also a block of binary data. A scheme like the DES (Data Encryption Standard) has a block size of 64 bits, using a 56-bit binary key, producing a 64 bit binary ciphertext. With 56-bit keys, there are 2^{56} (about 7.2 E+16) possible keys. Trying every possible key is known as the *brute force* attack.

When DES was released in 1975, that number of keys was effectively considered *infinite*. However, in 1998, a group at the Electronic Frontier Foundation built a machine called *Deep Crack* with extensive hardware parallelism that could try roughly a million DES keys every microsecond. At this rate, it took about 56 hours to try all possible keys.

RSA ran a "DES challenge" that involved decrypting a plaintext containing an English phrase in ASCII characters. There were several "crowd computing" attacks, which divided the entire key space into millions of chunks, which individuals could download and try to find the right key among. These typically took several months to locate the valid key, even with tens of thousands of people searching for it. With Deep Crack the problem was recognizing when it had found the right key. Most decryptions with a wrong key resulted in non-ASCII binary gibberish, which could be discarded. Another computer checked the relatively few decryptions that resulted in ASCII text to see if it contained any English words. If so, a human was alerted, who could determine if a valid English phrase was recovered. After Deep Crack, DES was considered no longer "strong". The assumption was that if a few people could create such a machine with a budget of about $200,000 that at least some governments (and certainly the U.S. NSA) probably already had and used such machines.



One of the circuit boards from Deep Crack, with multiple DES chips
(from Wikipedia Creative Commons)

Each additional bit of key length *doubles* the time that Deep Crack would take to find a key, so here is a table of how long it would take for various common key lengths:

Key Length	Time for Brute Force Attack with Deep Crack
56 bits	56 hours
64 bits	14,336 hours, or about 597 days
80 bits	107,000 years
128 bits	3.02 E+19 years
256 bits	1.02 E+58 years (longer than the heat death of the universe)

As you can see, with a symmetric key algorithm, the time required for a brute force attack quickly increases with increasing key length to totally impractical time scales. Even 128 bit keys are probably sufficient for most purposes. With 256 bit keys the time is greater than the expected life of the universe.

The most common symmetric key algorithm in use today is the AES (Advanced Encryption Standard) which replaced DES in 2000. It is very fast, and can use either 128 or 256 bit binary keys. There are currently no known attacks against AES that are faster than brute force. Several algorithms were entered in a competition to replace DES as the new standard, including TwoFish (successor to BlowFish) and Serpent. The algorithm that won (and is now known as AES) was called Rijndael (prounounced RAIN-dell). There are no royalties required for use, but many countries have laws restricting use of strong cryptography.

A.2.6 – Key Management

Whether you are using the "shift by n" scheme or AES, the sender must somehow securely communicate the symmetric key they used to the message recipient. This is not easy to do, as if it is sent via any plaintext channel (e.g. spoken over a telephone call, sent by fax, etc.), then the easiest way to attack the system is to intercept the key. Systems based on symmetric key cryptography alone have very complex key management systems, and these are usually the weakest link of any such system. Kerberos is an example of a cryptographic system based entirely on symmetric key cryptography (actually DES).

A.3 – Message Digest

Another cryptographic algorithm (message digest) does not include the entire contents of the input text in the output, even in scrambled form. It is not possible to recover the plaintext from a generated message digest. It is a *one way algorithm*. In fact, regardless of how large the input document is, the output is a fixed length (for a given message digest algorithm). For example, the MD5 message digest algorithm produces a 128 bit output, SHA1 produces a 160 bit output, and SHA-256 produces a 256 bit output. The output *characterizes* the input document. Any change at all in the input document is *amplified* by the message digest algorithm. Even a single bit difference in the input would result in a totally different output. This makes it very good for determining if two documents are identical (or the same document at two different points in time). If a document is modified in any way at all (changing, deleting or adding characters or bits), the generated message digest will be different after the modification. It is like an extremely sensitive checksum. The term Message Authentication Code (MAC) is

also sometimes used for Message Digest (not to be confused with the network term MAC, which refers to the Media Access Control layer, or MAC Addresses).

The following RFCs define Message Digest Algorithms:

- **RFC 1321, "The MD5 Message-Digest Algorithm", April 1992**
- **RFC 3174, "US Secure Hash Algorithm 1 (SHA1)", September 2001**
- RFC 3874, "A 224-bit One-way Hash Function: SHA-224", September 2004
- **RFC 4634, "US Secure Hash Algorithms (SHA and HMAC-SHA)", July 2006**

Note that RFC 4634 includes definition of the newer SHA-224, SHA-256, SHA-384 and SHA-512.

A.4 – Asymmetric Key Cryptography

In the early 1970s, some English cryptographers at GCHQ (James Ellis, Clifford Cocks and Malcolm Williamson) invented an entirely new type of cryptography, which was kept classified and not revealed until 1997. It was re-invented independently in 1976 by two researchers named Whitfield Diffie and Martin Hellman (based on work by Ralph Merkle). This scheme is called *asymmetric key cryptography*, and it involves using not *one* key, but a related *pair* of keys. If you encrypt something with one of the keys, only the other key of a pair will decrypt the ciphertext produced. Even the key used to encrypt the plaintext will not correctly decrypt the ciphertext. Later, three mathematicians from M.I.T. named Ron Rivest, Adi Shamir and Leonard Adleman created a company called RSA that was the first to commercially exploit this concept. Their asymmetric key algorithm (also called RSA) is still in extensive use today.

All asymmetric key algorithms depend on some very hard mathematical problem that is far faster to do in one direction (e.g. multiplying two large prime numbers together) than the other direction (e.g. factoring a giant product of two primes into the two prime numbers). The mathematical problems are referred to as *trap door functions* (because a trap door allows things to go easily in one direction, but not in the other). The fast direction (e.g. multiplying two giant primes) is used when you generate a pair of matched keys, but deriving the private key from the public key involves working the problem in the reverse direction (factoring the product of the two primes). The time required to do this increases greatly with the length of the product. A product that is 512 bits long (about 154 decimal digits) is considered weak. A product that is 1024 bits long (about 308 decimal digits) is considered fairly strong. A product that is 2048 bits (about 616 decimal digits) or longer is considered very strong. As computing power increases (and factoring theory improves) year by year, what was once considered strong is later considered weak, so key lengths must slowly increase. For example, cracking a 512 bit RSA key is estimated to take about 7,000 MIP-years to accomplish. In 1979, the Motorola 68000 ran at about 1 MIPS, so it would take 7,000 years for one machine (or 70 years for 100 machines) to break one 512 bit key. In 2000, the AMD Athlon ran at about 3500 MIPS, so it would take about 2 years to do the same thing. The most recent Intel Core i7 (i980EE) runs at about 147,000 MIPS, so one machine with this CPU would be able to crack a 512 bit key in about 18 days. Such increases will eventually result in having to use impractically long keys with RSA. Fortunately, a newer asymmetric key algorithm called Elliptic Curve Cryptography (ECC) has better characteristics (strength goes up more rapidly with increasing key length

than it does with RSA). At some point, as computing power increases, cryptographic systems will have to switch over to using ECC. The standards already support it.

There is also an asymmetric key algorithm called the Digital Signature Algorithm (DSA), created by the U.S. NSA (National Security Agency). DSA is part of the Digital Signature Standard (DSS) which is specified in FIPS 186. The DSS was adopted in 1996 as FIPS 186-1, and then expanded in 2000 as FIPS 186-2, and again in 2009 as FIPS 186-3. There is no general encryption algorithm (e.g. for digital envelope) associated with DSA – it is just for digital signatures.

Let's say Alice creates two keys. She publishes one of them (called her *public* key) to the entire world. The other one (called her *private* key), she never shows to anyone. Anyone in the world can obtain and use her public key to encrypt things, but she is the only one than can decrypt things encrypted with her public key. So if she wanted to send something securely to Bob, she would obtain Bob's public key (even through insecure channels – it does not matter who sees it). She would encrypt her message using Bob's public key, and send the encrypted message to him. Anyone in the world would be able to send Bob such a message, because all they need is his public key. He would use his private key to decrypt it. He is the only person in the world who could do this (assuming no one has discovered his private key). This provides complete privacy, without having to do complex and insecure symmetric key distribution.

A.4.1 – Digital Envelopes

Compared with symmetric key cryptography, asymmetric key cryptography is *very* slow (in terms of how many bytes per second can be encrypted), so it is not commonly used to encrypt entire long messages. It is usually used just to encrypt a short (e.g. 128 bit) symmetric session key or a short (e.g. 160 bit) message digest. So, in a real system (such as S/MIME E-mail), Alice would generate a random symmetric session key (just for this E-mail message, never to be used again), encrypt the message with it (using AES), then encrypt the symmetric session key and Bob's public key (using RSA). She would send both the encrypted message and the encrypted session key to Bob. He would first decrypt the session key with his private key (using RSA), then use the recovered session key to decrypt the message (using AES). This is called creating a *digital envelope*. A digital envelope provides *privacy*. Asymmetric key cryptography solves the problem of how to securely distribute symmetric keys.

A.4.2 – Digital Signatures

Asymmetric key cryptography is also useful for creating digital signatures, when used in combination with a message digest algorithm. As an example, to produce a digital signature of a document, Alice first produces a message digest of the message (using SHA1) then encrypt the message digest with her private key (using RSA). The result would be encoded into ASCII and appended to the message. She would send the message including the digital signature to Bob. Bob would strip off the digital signature, and produce a *new* message digest of the message (using SHA1). He would also decrypt the encrypted message digest with Alice's public key (using RSA). He would then compare the recovered message digest with the newly generated message digest. If they match, then the digital signature is *valid*. If they don't match then either the message was signed using someone else's private key (hence was probably sent by someone else), or the message had been tampered with along the way. In either case, the message should not be accepted as valid. A digital signature provides *sender authentication* (knowing

for certain that it was sent by the person it appears to be from), and *message integrity* (being able to detect any possible change to the message). These are two very useful things to know about a message.

A.4.3 – Combined Digital Signature and Digital Envelope

By itself, a digital signature does not provide *privacy*. You may not care who *reads* a message, but you want to know for certain who it came from, and whether or not it has been tampered with along the way. In this case, only a digital signature is needed. If you *also* want privacy, then you must both digitally sign and then digitally envelope the message. There is no conflict between the two processes. S/MIME E-mail allows you to do neither, either or both. They are completely independent processes (although if combined, the digital signature must be done first and checked last).

A.4.4 – Public Key Management and Digital Certificates

When Alice sends Bob a digitally signed message, she has all the keys she needs (her own private key) at the time she composes the message. When he is reading the message, Bob needs Alice's public key, which she can simply append to the message. If Alice appended her key, Bob can use the appended public key to check the signature. If she didn't, Bob can obtain her public key in various other ways, perhaps she published her key on a website via HTTP, or in an LDAP directory.

When Alice sends Bob a digitally enveloped message, she needs Bob's public key, which she may not have. Bob might have published it on his website, or in an LDAP directory, or he could simply send Alice a digitally signed message first. Many secure E-mail systems save the public keys of the senders of all digitally signed messages in case you want to later send them digitally enveloped messages. When Bob receives a digitally enveloped message, he has all the keys he needs to open it (his own private key).

Regardless of how Bob obtained Alice's public key, how does he know for certain that the key really belongs to Alice? In general, anytime you use a public key, you must determine if it is the real public key for that user. If Charlie can trick Bob into using *his* public key to check what he thinks is Alice's signature, Charlie can sign the message using his own private key and Bob will think it is a valid signature from Alice (this is called the *public key substitution threat*).

A public key must be distributed embedded in a digitally signed document that includes other identifying information, such as the key owner's name, email address, expiration date, etc. This is called a *public key digital certificate*. It should be produced and signed by a *trusted third party* (someone other than Alice and Bob, but whom they both trust). For example, VeriSign will produce digital certificates for anyone. Such an organization is called a *Certification Authority* or CA for short. They must verify all of the information to be embedded in the digital certificate as correct, and operate in a secure manner to prevent bogus certificates from being issued. The digital signature *binds* all of the included information together.

The *trust domain* for a given application consists of all the involved parties. For a bank and their customers, the bank could issue digital certificates to their customers and supply them with the necessary root certificates to install in their browsers (this is called a *private hierarchy PKI*). If the

involved parties can be just anyone in the world, then the trusted third party must be someone like VeriSign, who have embedded their root certificate in all browsers (this is called a *public hierarchy PKI*).

A.4.4.1 – Chain of Trust

Since each person's digital certificate is signed by a Certification Authority (using their own private key), you must verify the signature in each person's digital certificate by using the CA's public key. How do you know that you have the real public key for the CA, and not that of some hacker? The necessary public key is provided in *yet another* digital certificate (an *intermediate certificate*). The intermediate certificate is signed by the CA's *root* private key, and you verify the intermediate certificate's digital signature with the CA's root public key. The root public key is provided to you in a self-signed certificate (one signed by the root private key), but is installed in your application (e.g. your browser) by the vendor so you can trust it. Verifying all of the certificates from that of the person you are communicating with, up to the root certificate of the relevant Certification Authority is called *climbing a chain of trust*.

A.4.5.2 – Certificate Validity Period

Like a credit card, a digital certificate has an expiration date, past which it should not be accepted by anyone. Certificates are usually renewed (re-issued) once a year, after re-verifying all the embedded information. This insures that the embedded information has been verified recently. Unlike credit cards, digital certificates also have a starting date. The certificate is only valid from the starting date until the expiration date.

A.4.5.3 – Certificate Revocation

When you use your credit card in a store, they used to check its number against a printed book of *revoked* card numbers (from people who didn't pay their credit card bill). If your number was in the book, the store would not accept your card, and would usually cut it up. Later this was automated, so that the number (and the amount of the transaction) was sent online to the credit card company, who determined if the card was still valid, and the amount was within their remaining credit limit for that card.

Digital Certificates can also be revoked before their expiration date. There are two mechanisms for doing this: *Certificate Revocation Lists* and *Online Certificate Status Protocol (OCSP)*. The first scheme (CRL) is the older, and is similar to the printed books of revoked credit cards. CRLs are published (via HTTPS) periodically (usually daily), so there may be some delay before everyone can become aware that a certificate has been revoked. The other scheme (OCSP) is more recent, but allows users to determine the current status of certificates instantly. Not all client applications support OCSP.

A.4.5.4 – Public Key Infrastructure (PKI)

It is the responsibility of any client or server that uses digital certificates to verify all of these things before accepting a certificate as valid (for an example, see the details of the SSL/TLS handshake later in

this chapter). The issuance of digital certificates, and all of the things related to the management of them, is called a *Public Key Infrastructure* (PKI). Creating and running a PKI is not a trivial thing, but it is necessary for any system that uses asymmetric key cryptography.

A.5 – Hash-Based Message Authentication Code (HMAC)

There are also some message digest algorithms that use a key, similar to an encryption algorithm. They still produce a fixed length message digest, like a regular message digest, but a key is used in the process, which modifies the way the digest is created. For a given input text, and a given HMAC algorithm, if you change the key, you will get a very different message digest. This is kind of a lightweight digital signature (no asymmetric cryptography is used). HMAC is used in IPsec for the Authentication Header. The sender produces an HMAC of the relevant fields of the packet, using a given key. One key is shared securely with the recipient for a lot of packets, via either "shared secret key" or via IKE. Upon receipt of a packet, the recipient calculates the HMAC of the same relevant fields, using the HMAC key that was used by the sender. If an identical message digest is produced, then this is proof that the relevant fields of the packet have not been tampered with, and could only have been sent by the purported sender (not by someone else).

Using HMAC is *much* faster than doing a full digital signature, which is why it is "lightweight". A full digital signature would be more secure, and far more difficult to fake, but the computing overhead on every packet would be enormous, and packet throughput would drop dramatically. Likewise for ESP, a full digital envelope is not used for every packet, but only symmetric key encryption is used, with a key that is exchanged once (for a large number of packets) using either "shared secret" or IKE.

Relevant standards for Hash-Based Message Authentication Codes are:

- **FIPS PUB 198, The Keyed-Hash Message Authentication Code, March 2002.**
- **RFC 2104, "HMAC: Keyed-Hashing for Message Authentication", February 1997.** This defines several algorithms, including HMAC-MD5, HMAC-SHA1 and HMAC-RIPEMD (based on the MD5, SHA1 and RIPEMD message digests, respectively).
- **RFC 2403, "The Use of HMAC-MD5-96 within ESP and AH", November 1998.**
- **RFC 2857, "The Use of HMAC-RIPEMD-160-96 within ESP and AH", June 2000.**
- RFC 4634, "US Secure Hash Algorithms (SHA and HMAC-SHA)", July 2006.
- **RFC 4868, "Using HMAC-SHA-256, HMAC-SHA-384 and HMAC-SHA-512 with IPsec", May 2007.**

A.6 – Internet Key Exchange (IKE)

Internet Key Exchange is part of IPsec. It is used to automate the distribution of symmetric session keys to any number of IPsec enabled nodes, in a secure manner. It depends heavily on asymmetric key cryptography. IKE is based on the Diffie-Hellman Key Agreement algorithm. It is covered in more detail in the chapter on TCP/IPv6 Advanced Protocols.

Relevant standards are:

- **RFC 2409, "The Internet Key Exchange (IKE)", November 1998**
- **RFC 4306, "Internet Key Exchange (IKEv2) Protocol", December 2005**

A.6.1 – IKE using IPsec Digital Certificates

If IKE is used to automate IPsec key management, a full PKI must still be deployed, to create and manage the IPsec digital certificates. One certificate must be generated for each node that participates in IPsec traffic. With IPsec certificates, the public key is *bound* to IP address(es), not to a Fully Qualified Domain Name and an organization name (as in server certificates) or to a person's name and E-mail address (as in client certificates). To simplify the process of having every IPsec enabled node obtain the necessary IPsec digital certificate, the Simple Certificate Enrollment Protocol (SCEP) is typically used to allow nodes to request and obtain IPsec certificates from a CA without the complex and time consuming manual processes used to obtain server or client certificates.

The shared secret key distribution mechanism is widely used with current implementations of IPsec , but it is likely that only IPsec certificate based automated key exchange with IKE will be allowed in the future.

Relevant standards for IPsec Digital Certificates in IKE are:

- **RFC 4809, "Requirements for an IPsec Certificate Management Profile", February 2007.**
- **RFC 5406, "Guidelines for Specifying the Use of IPsec Version 2", February 2009.**

A.6.2 – Diffie-Hellman Key Exchange

One of the first applications of asymmetric cryptography (once it was re-invented) was to securely exchange a symmetric session key between two communicating parties. This is still used today as the heart of the Internet Key Exchange (IKE).

Let's use Diffie-Hellman to exchange a symmetric session key between Alice and Bob. First there are two *public parameters* "p" and "g". The "p" parameter is a very large prime number, while "g" is an integer less than p with certain characteristics. Anyone can know these values. They can be agreed to and shared well in advance, even over insecure channels. They can even be defined and built into a cryptographic system, such as IKE.

1. Alice generates a random private value "a", and Bob independently generates another random private value "b". They don't share these values with anyone, even each other.
2. Alice generates her *public value* $X = g^a \bmod p$, and Bob generates his *public value* $Y = g^b \bmod p$.
3. Alice sends her public value (X) to Bob, and Bob sends his public value(Y) to Alice, over an insecure channel.
4. Alice computes $k_{ab} = (Y)^a \bmod p$, and Bob computes $k_{ba} = (X)^b \bmod p$.
5. Remarkably, $k_{ab} = k_{ba}$, so Alice and Bob now have a shared secret value, from which a symmetric session key can be derived.

It is computationally infeasible to try to derive the shared secret given Alice's and Bob's *public values* (which are sent in the clear) due to the *discrete logarithm problem* (another trap door function), assuming p is sufficiently large.

A.7 – Secure Socket Layer (SSL) / Transport Layer Security (TLS)

Probably the most popular cryptographic application is SSL/TLS, which is used to secure HTTP (web) connections. It was originally created by Netscape in their web server and web browser. SSL 2.0 implemented server to client authentication and symmetric key exchange. In a non-secure connection (e.g. web over port 80), a client connects to the server, and is granted access. No authentication is performed, no symmetric key is exchanged, and traffic is exchanged between the server and the client in plaintext.

SSL is implemented as a *shim layer* between the application layer and the Transport Layer. Only TCP is supported in the Transport Layer – there is no SSL or TLS for UDP. The application must have knowledge of SSL and call socket I/O routines in the SSL library to create a socket, to read and write data, and to close the socket (instead of directly calling the usual routines in the Transport Layer). Significant changes are required to the source code of both the server application and the client application.

SSL can be used in almost any query/response client-server protocol which is implemented over TCP. With SSL, there is both a non-secure port and a corresponding secure port for each protocol. It has been successfully used in HTTP (80/443), LDAP (389/689), POP3 (110/995), IMAP (143/993) and SMTP (25/465). A commercial toolkit from RSA (BSAFE), or an open source toolkit (OpenSSL) can be used to add SSL to any client-server network application.

A server certificate must be obtained and installed in the server application. This digital certificate binds a Fully Qualified Domain Name and an organization name to the public key. The root certificate of the server certificate's hierarchy must be installed in both the server application and the client application, along with any necessary intermediate certificate(s). These might be hard wired into the applications by the vendor, or installed manually by a system administrator.

If a connection is attempted via the secure port (e.g. web over port 443), then the following exchange takes place before the normal web traffic continues:

1. The server sends its digital certificate to the client in plaintext.
2. The client verifies the validity of the server certificate as follows:
    a. The digital signature of the server certificate must be verified using the public key intermediate certificate of the signer (e.g. VeriSign).
    b. The digital signature of the intermediate certificate must be verified using the public key root certificate of the server certificate signer (e.g. VeriSign), as hardwired in the web browser by the vendor.
    c. The current time must be within the validity period of the certificate (after the starting date, but before the expiration date).
    d. The Fully Qualified Domain Name from the browser URL must match the Fully Qualified Domain Name (common name) in the server certificate.

    e. The server certificate's revocation status must be checked either by consulting the appropriate Certificate Revocation List for the server certificate, or via OCSP.
3. The client and the server negotiate the strongest cryptographic algorithms that they hold in common, both for asymmetric key cryptography (e.g. RSA 1024 bit) and asymmetric key (e.g. RC4 128 bit).
4. The client generates a challenge string, encrypts it with the public key from the server certificate using the agreed upon asymmetric key algorithm, and sends the encrypted string to the server.
5. The server decrypts the encrypted challenge string using its own private key and the agreed upon asymmetric key algorithm. It then returns the result to the client, where it must match the original challenge string which the client had created. This establishes server to client authentication. Only the owner of the private key associated with the public key from the server certificate would be able to pass this cryptographic challenge.

    [optional client authentication handshake in SSL 3.0 goes here – see below]

10. The client generates a symmetric session key for the agreed upon symmetric key algorithm and key length. It encrypts this symmetric key using the server's public key, and the agreed upon asymmetric key algorithm. The encrypted key is sent to the server.
11. The server receives the encrypted symmetric session key and decrypts it using its private key and the agreed upon asymmetric key algorithm. Now the client and the server share a common symmetric session key, which has been exchanged securely. Even if this entire exchange is intercepted by a third party, the symmetric session key cannot be discovered.

Following these steps, all traffic between client and server is encrypted. The sender encrypts the traffic using the shared symmetric session key, and the recipient decrypts the traffic using the same key. Periodically (based on time and or traffic volume) a new symmetric session key will be negotiated and used.

A.7.1 – Secure Socket Layer 3.0 – Optional Strong Client Authentication

In SSL 3.0 (1996), Netscape added an optional second part of the cryptographic handshake. It is enabled by selecting strong client authentication on the server. This can be optional (strong client authentication *allowed*), in which case if the client has no acceptable client certificate the server will fall back to username/password authentication. It can also be mandatory (strong client authentication *required*). It this case, if the client has no acceptable client certificate, it will not be able to connect to the server.

The root certificate of the client certificate hierarchy (plus any intermediate certificates) must be installed in both the server and client applications, either hardwired in the application by the vendor, or installed manually by the administrator.

After step 5 above, if strong client authentication is enabled, the handshake includes the following new steps:

6. The server sends a request to the client to present a client certificate, including a filter that specifies criteria for acceptable client certificates.

7. The client presents a list of all acceptable client certificates found in its certificate database, from which the user selects one. That certificate is sent to the server in plaintext.
    a. The digital signature of the client certificate must be verified using the public key intermediate certificate of the signer (e.g. VeriSign).
    b. The digital signature of the intermediate certificate must be verified using the public key root certificate of the client certificate signer (e.g. VeriSign), as hardwired in the web browser by the vendor (or installed by the user).
    c. The current time must be within the validity period in the client certificate (after the starting date, but before the expiration date).
    d. The common name from the client certificate is extracted to use as the login name.
    e. The server certificate's revocation status must be checked either by consulting the appropriate Certificate Revocation List for the server certificate, or via OCSP.
8. The server generates a challenge string, encrypts it with the public key from the client certificate using the agreed upon asymmetric key algorithm, and sends the encrypted string to the client.
9. The client decrypts the encrypted challenge string using its own private key and the agreed upon asymmetric key algorithm. It then returns the result to the server, where it must match the original challenge string which the server had created. This establishes client to server authentication. Only the owner of the private key associated with the public key from the client certificate would be able to pass this cryptographic challenge. At no point did the owner of the private key have to expose it to anyone in order to prove possession of it.

A.7.2 – Transport Layer Security (TLS) – Continuation of SSL as an IETF Standard

When Netscape released their proprietary SSL 3.0 standard to the IETF, it was further refined and extended as TLS 1.0. Additional cryptographic suites were installed, including Diffie-Hellman asymmetric key algorithms and additional symmetric key algorithms. Also, an optional handshake mechanism was defined for those protocols that could support it (e.g. SMTP) which allows both secure and non-secure connections to use the same port. A connection over TLS begins in the non-secure state, but a negotiation (e.g. using ESMTP extension announcements and commands) can initiate the SSL/TLS handshake and switch to a secure connection over the same port.

There are really no RFCs concerning SSL, as it was a Netscape proprietary protocol. Here are some of the standards related to TLS:

- **RFC 2246, "The TLS Protocol Version 1.0", January 1999**
- **RFC 2487, "SMTP Service Extensions for Secure SMTP over TLS", January 1999**
- **RFC 2595, "Using TLS with IMAP, POP3 and ACAP", June 1999**
- **RFC 2817, "Upgrading to TLS Within HTTP/1.1", May 2000**
- RFC 3546, "Transport Layer Security (TLS) Extensions", June 2003
- RFC 4217, "Securing FTP with TLS", October 2005
- **RFC 4346, "The Transport Layer Security (TLS) Protocol Version 1.1", April 2006**
- **RFC 5246, "The Transport Layer Security (TLS) Protocol Version 1.2", August 2008**

A.7.3 – Link Oriented Nature of SSL/TLS

SSL/TLS is inherently a link-oriented protocol. It can only secure a single connection such as web browser to web server. For multi-connection scenarios, such as E-mail (client to local server, local server to remote server and remote client to remote server), there is no simple way to insure that all of the links will use SSL/TLS protection, but even if they do, the traffic will be in plaintext on each intervening node, e.g. on intermediate mail servers. S/MIME is the preferred technology for securing E-mail. It takes place in the sending and receiving client, and is transparent to any intervening servers. This is called “end to end”.

A.7.4 – SSL-VPN

VPNs typically need to be end-to-end as well, and IPsec is. Unfortunately IPsec does not work well on IPv4 networks due to the omnipresence of NAT. Because of this, on the First Internet, many people now use a VPN scheme based on SSL, called “SSL-VPN”. There is no IETF standard for this, it uses an Application Layer technology (SSL) in the Internet Layer, and is inherently link-oriented. The only alternative is to add NAT traversal technology to all IPsec clients and servers, which greatly complicates them and introduces serious security issues. VPNs will work well using the approved technology (IPsec) without any NAT traversal mechanisms, only once IPv6 connections are available between the nodes to be connected with VPNs.

Bibliography

TCP/IPv4

Comer, Douglas E. Internetworking with TCP/IP, Volume 1 (5th Ed.). Prentice Hall, 2005.

Comer, Douglas E. Internetworking with TCP/IP, Volume II: ANCI C Version: Design, Implementation and Internals (3rd Ed.). Prentice Hall, 1998.

Comer, Douglas E.; Stevens, David L. Internetworking with TCP/IP Volume III: Client-Server Programming and Applications – BSD Socket Version (2nd Ed.). Prentice Hall, 1996.

Comer, Douglas E.; Stevens, David L. Internetworking with TCP/IP, Volume III: Client-Server Programming and Applications, Linux/Posix Sockets Version. Prentice Hall, 2000.

Comer, Douglas E.; Stevens, David L. Internetworking with TCP/IP Volume III: Client-Server Programming and Applications – Windows Sockets Version. Prentice Hall, 1997.

Doyle, Jeff; Carroll, Jennifer. Routing TCP/IP, Volume 1 (2nd Ed.). Cisco Press, 2005.

Doyle, Jeff; Carroll, Jennifer DeHaven. Routing TCP/IP, Volume II (CCIE Professional Development). Cisco Press, 2001.

Stevens, W. Richard. TCP/IP Illustrated, Volume 1: The Protocols. Addison-Wesley Professional, 1994.

Wright, Gary R.; Stevens, W. Richard. TCP/IP Illustrated, Volume 2: The Implementation. Addison-Wesley Professional, 1995.

Stevens, W. Richard. TCP/IP Illustrated, Volume 3: TCP for Transactions, HTTP, NNTP, and the UNIX Domain Protocols. Addison-Wesley Professional, 1996.

TCP/IPv6

Blanchet, Marc. Migrating to IPv6: A Practical Guide to Implementing IPv6 in Mobile and Fixed Networks. Wiley, 2006.

Davies, Jospeh. Understanding IPv6 (2nd Ed.). Microsoft Press, 2008.

Desmeules, Regis. Cisco Self-Study: Implementing Cisco IPv6 Networks (IPv6). Cisco Press, 2003.

Grossetete, Patrick; Popviciu, Ciprian P.; Wettling, Fred. Global IPv6 Strategies: From Business Analysis to Operational Planning. Cisco Press, 2008.

Hagen, Silvia. IPv6 Essentials. O'Reilly Media, Inc., 2006.

Hagino, Jun-ichiro itojun. IPv6 network Programming. Digital Press, 2004.

Hogg, Scott; Vyncke, Eric. IPv6 Security. Cisco Press, 2008.

Huitema, Christian. IPv6: The New Internet Protocol (2nd Ed.) Prentice Hall PTR, 1998.

Li, Qing; Jinmei, Tatuya; Shima, Keiichi. IPv6 Core Protocols Implementation. Morgan Kaufmann, 2006.

Li, Qing; Jinmei, Tatuya; Shima, Keiichi. IPv6 Advanced Protocols Implementation. Morgan Kaufmann, 2007.

Li, Qing; Jinmei, Tatuya; Shima, Keiichi. IPv6 Socket API Extensions: A Programmer's Guide. Morgan Kaufmann, 2009.

Li, Qing; Jinmei, Tatuya; Shima, Keiichi. Mobile IPv6: Protocols and Implementation. Morgan Kaufmann, 2009.

Malone, David; Murphy, Niall Richard. IPv6 Network Administration. O'Reilly Media, 2005.

Minoli, Daniel. Voice over IPv6: Architectures for Next Generation VoIP Networks. Newnes, 2006.

Popoviciu, Ciprian; Levy-Abegnoli, Eric; Grossetete, Patrick. Deploying IPv6 Networks. Cisco Press, 2006.

Siil, Karl A. IPv6 Mandates: Choosing a Transition Strategy, Preparing Transition Plans, and Executing the Migration of a network to IPv6. Wiley, 2008.

Soliman, Hesham. Mobile IPv6: Mobility in a Wireless Internet. Addison-Wesley Professional, 2004.

Stockebrand, Benedikt. IPv6 in Practice: A Unixer's Guide to the Next Generation Internet. Springer Berlin Heidelberg, 2009.

van Beijnum, Iljitsch. Running IPv6. Springer-Verlag, 2006.

DNS/DHCP

Liu, Cricket; Albitz, Paul. DNS and BIND (5th Ed.). O'Reilly Media, 2006.

Aitchison, Ron. Pro DNS and BIND. Apress, 2005.

Liu, Cricket. DNS & BIND Cookbook. O'Reilly Media, 2002.

Droms, Ralph; Lemon, Ted. DHCP Handbook, 2nd Edition. Sams, 2002.

IPsec

Bollapragada, Vijay; Khalid, Mohamed; Wainner, Scott. IPSec VPN Design. Cisco Press, 2005.

Carmouche, James Henry. IPsec Virtual Private Network Fundamentals. Cisco Press, 2006.

Davis, Carlton. IPSec Securing VPNs. McGraw-Hill / OsborneMedia, 2001.

BSD & Linux Networking

Bautts, Tony; Dawson, Terry; Purdy, Gregor N. Linux Network Administrator's Guide. O'Reilly Media, 2005.

Doraswamy, Naganand; Harkind, Dan. IPsec (2nd Ed.). Prentice Hall, 2003.

Hansteen, Peter N.M. The Book of PF: A No-Nonsense Guide to the OpenBSD Firewall. No Starch Press, 2008.

Hong, Bryan J. Building a Server with FreeBSD 7. No Starch Press, 2008.

Lucas, Michael W. Absolute FreeBSD: The Complete Guide to FreeBSD, 2nd Edition. No Starch Press, 2007.

Lucas, Michael. Absolute OpenBSD: UNIX for the Practical Paranoid. No Starch Press, 2003.

McKusick, Marshall Kirk; Neville-Neil, George. The Design and Implementation of the FreeBSD Operating System. Addison-Wesley Professional, 2004.

Palmer, Brandon; Nazario, Jose. Secure Architectures with OpenBSD. Addison-Wesley Professional, 2004.

Rash, Michael. Linux Firewalls: Attack Detection and Response with iptables, psad and fwsnort. No Starch Press, 2007.

Schroder, Carla. Linux Networking Cookbook. O'Reilly Media, 2007.

CORPORATE PROFILE

THE COMPANY

InfoWeapons can be summed up in one word -- **Solid**.

InfoWeapons is a six-year-old network appliances company marketing its products to enterprises, governments and large service providers. Its appliances allow for rapid and cost-effective migration to IPv6, the next generation Internet protocol, and also provide support for IPv4, the current Internet standard. The company's "Solid" suite of products incorporates "defense in-depth", an approach to information and electronic security conceived by the NSA.

InfoWeapons has an experienced management team with complementary industry sector expertise in IT Security, Network Infrastructure and Voice and Data Communications. Several of its executives have successfully founded and sold other enterprises.

InfoWeapons management's diverse expertise culminating from decades of experience with VeriSign and CipherTrust (Secure Computing) places the team in an excellent position to take advantage of the opportunities presented by the transition of networks and the Internet to IPv6.

Company Facts
- Corporation established in 2004
- Offices – North America and Asia

Management Team
- Lawrence Hughes, Chairman & CTO
- Luis Gopez, CEO

Advisors
- Morris, Manning and Martin LLP - Legal
- Bennett Thrasher PC - Auditor
- Hood Marketing Solutions - Marketing
- Direct Communication Specialists - Call Center

Strategic/OEM Partners
- IBM
- Dell
- Supermicro
- pbxnsip & snom

PRODUCTS

In just a short period of time, InfoWeapons has created and launched **SolidDNS™**, **SolidPBX™** and **SolidWall™**, products that deliver rock-solid security, uncompromising performance, and unparalleled ease of use.

SolidDNS™ is a robust DNS appliance designed for the next-generation network, but capable of bridging legacy networks with IPv6. It provides reliable and secure DNS service whether deployed in small offices or by service providers in the cutting-edge of technology. It offers support for new IP-based services such as ENUM, VoIP, IPTV and is the first appliance with a graphical interface for DNSSEC and DHCPv6.

SolidDNS™ carries the IPv6 Ready Phase-2 "Gold" Logo (Logo ID: 02-C-000272), which certifies that the appliance supports core IPv6 protocols, has passed the tests for functionalities such as mobility and remote administration, and is ready for commercial deployment.

SolidPBX™ is an advanced Voice over IP (VoIP) private branch exchange (PBX) appliance for the enterprise. It offers numerous telephony, PBX, security, and contact management features over IPv4, IPv6 or dual-stack networks. It boasts an expanding repertoire of hardphones, softphones, and other communications hardware. The SolidPBX™ 2.0 features VLAN tagging, high availability with expanded IPv6 capabilities, extended FXO gateway support, and highlights enhanced web management with more easy to use wizard-driven configurations. A Hospitality Edition that features Property Management Systems (PMS) integration and hospitality-specific prompts is also in the pipeline.

SolidWall™ is an easy-to-use network security and IPv6 tunnel endpoint appliance. It combines the functions of a dual-stack, stateful perimeter and internal network firewall, Quality of Service (QoS) capabilities, VPN endpoint, and IPv6 tunnel connectivity. The appliance is designed to enhance network security while allowing organizations to set up IPv6 connectivity even over IPv4 service provider connections, quickly and safely bringing IPv6 into their networks.

The Quickstart Kit is a bundle of IPv6 products and services designed to help organizations deploy IPv6 quickly and easily. The Basic Kit consists of SolidDNS™ SFF-Series, SolidWall™ Lite, easy-to-use IPv6 wizards and IPv6 training to help get organizations on the right track and stay the course. Also available is Quickstart Kit Plus, which is the basic kit plus SolidPBX™ Office 20. InfoWeapons engineers are also available to assist progressive organizations who have the foresight to deploy IPv6 now.

SolidDNS

THE SolidDNS™ SERVER APPLIANCE PROVIDES SECURE, DUAL-STACK DNS SERVICE TO NETWORKS OF ANY SIZE

SolidDNS™ for the Next Generation Network

As enterprises begin the transition into IPv6, they face the daunting challenge of adapting existing tools to match the specifications of the new standard. Nowhere will this be more apparent than in the basic infrastructure services such as network configuration, security, and naming services.

The SolidDNS™ appliances are designed to address the critical need for domain name services in an IPv6 environment. The appliance integrates the latest security patches for BIND, including source port randomization to counter recently discovered DNS flaws, while providing backward compatibility with existing IPv4 networks.

The SolidDNS™ secure DNS and DHCP appliance platform from InfoWeapons boasts extensive support for DNSSEC with NSEC3, VLAN tagging, IP aliasing, and DNAME. Using graphical management interfaces, the appliance provides tools for managing cryptographic keys and features bulk zone signing, automatic zone re-signing and key rollover, user-specified key lifetimes, and can act as an authoritative DNSSEC server.

The latest SolidDNS™ 4.1.614 version allows you to export SolidDNS™ packets into the Netflow™ Service to detect anomalous or abnormal activity against the appliance. It also supports "blackhole" statements to ignore fake queries coming from a specified IP address or network and allows the wildcard asterisk "*" for CNAME to resolve even misspelled queries. Moreover, the latest version provides the option to flush the cache of a specific domain and download the cache when it reaches 3MB.

SolidDNS™ Service Provider Series

The SolidDNS™ Service Provider Series appliance platform, which focuses on high performance and reliability, is for telecommunication companies, service providers and large enterprises. The hardware is redundant with swappable power supplies and disks designed for high reliability. Systems CPUs and memory are optimized for speed and scalability. These appliances can be deployed in stand alone or in high availability modes, and take on various roles from internal caching nameservers in high-demand environments to external nameservers for hundreds of busy domains and addresses.

With predictable and stable performance of 125,000 queries per second, the SolidDNS™ Service Provider Series appliances are some of the fastest in their class. It comes fully equipped with a rack-mountable 1U full-length form factor, Quad-Core 64-bit CPU, 8 to 64 GB RAM, 2xGigE NIC, 2x250 GB mirrored disks, and redundant power. Single D/C (Direct Current) power supply variant is also readily available.

SolidDNS™ Enterprise Series

The SolidDNS™ Enterprise Series appliances are ideal for medium-sized enterprises. Everything an organization needs to operate a secure DNS and DHCP server is included in the Enterprise Series. With software functions to securely separate different networks and easy-to-use management interfaces, a SolidDNS™ Enterprise Series appliance can function as an internal and external DNS server at the same time. A caching server model is also available to improve network performance.

The SolidDNS™ Enterprise Series appliances, which have a predictable high performance of 100,000 queries and 600 leases per second, can be deployed in stand alone or in high availability modes. They come in rack-mountable 1U half-length form factor, Quad-Core 64-bit CPU, 4 to 8 GB RAM, 2xGigE NIC, 160 GB HD, and in auto-switching universal voltage.

SolidDNS™ Xpress Series

The SolidDNS™ Xpress Series appliances are intended for small businesses and branch offices. Caching and Premium models are available to meet different performance requirements. The Premium model has the full DNS and DHCP feature set for both IPv4 and IPv6 networks. Both models come in rack-mountable 1U half-length form factor, Dual-Core 64-bit CPU, standard 2GB RAM upgradeable, 2xGigE NIC, 160GB HD, and in auto-switching universal voltage.

Performance of the Xpress Series appliances is at 60,000 queries per second, a 50% increase in the latest software version.

SolidDNS™ Small Form Factor Series

The SolidDNS™ Small Form Factor (SFF) Series appliances are designed and priced for small and home offices. Despite being entry-level models, the Standard Edition has the full DNS feature set for both IPv4 and IPv6 networks.

These appliances come in Small Form Factor (SFF), Single-Core 64-bit, 1GB or 2GB RAM, 1x10/100 or 2xGigE NIC,160GB HD, and in universal voltage. The SolidDNS™ SFF Series appliances have a performance of 11,000 queries per second.

SolidDNS™ Softpliance

The SolidDNS™ Softpliance brings together the best features from the appliance approach and delivers it in a software format ready to run in a VM-Ware environment. It provides cost-effective DNS and DHCP services to IPv4, IPv6 or dual-stack networks.

The SolidDNS™ Softpliance eliminates complex, expensive installation while allowing flexibility in the selection of hardware vendor. A new virtual appliance can be put into production use in a matter of minutes and can be scaled up or down to deliver the type of service your business requires.

SolidPBX

THE SolidPBX™ IS A DUAL-STACK IP-PBX APPLIANCE FOR THE ENTERPRISE.

SolidPBX™ Advanced VoIP Appliance

SolidPBX™ brings the cost-effectiveness and flexibility of Voice over IP (VoIP) technology to the enterprise telephone exchange. SolidPBX™ is a VoIP private branch exchange (PBX) appliance that has numerous telephony, PBX, security, and contact management features, and can be managed using flexible user interfaces. SolidPBX™ operates on IPv4, IPv6 and dual-stack networks and has support for an expanding repertoire of hardphones, softphones, and other communications hardware.

The appliance is designed with numerous security features built-in. With upgrade and after-sales support, SolidPBX™ is a reliable investment for the future of your organization's communication needs.

The SolidPBX™ 2.0 features VLAN tagging, high availability with expanded IPv6 capabilities, extended FXO gateway support, and highlights enhanced web management with more easy to use wizard-driven configurations.

Office Edition

The SolidPBX™ Office Edition is the base model for this product series. With support for a growing repertoire of soft and hard phones, communications protocols, and advanced PBX call management features, it is designed to meet the IP (Internet Protocol) telephony needs of small to large businesses. A Small Form Factor (SFF) model is also available with 20 extensions designed for small and home offices.

Licenses are available in 20, 50, 80, 100, 120, and 150 extensions.

Call Center Edition

The Call Center Edition of SolidPBX™ provides a powerful and dependable contact center platform for businesses of all sizes. It offers all the standard features of the SolidPBX™ Office Edition with the addition of agent and call management features for contact centers.

Licenses for 50, 100, 150 extensions are available.

Standard Software Features

Advanced Network Features

- IPv4
- IPv6
- Dual Stack (IPv4 and IPv6)

Security

- Certificate Management
- Strong Authentication of administrators
- TLS and SRTP support
- Loadable SSL/TLS certificate for VoIP
- SSL Encryption
- Password and PIN per extension
- Built-in firewall
- Built-in stack protection
- Jail separated process

Mobility Support

- Call forwarding to cell phone
- Voicemail triggers call to cell phone
- Inbound call cell phone detection
- Camp on from cell phone
- Hot desking

Convergence

- VoIP
- Conferencing
  - Conference mixer
  - Instant conference
  - Conference scheduler with e-mail invitation
- Voicemail
  - Private and shared voicemail
  - Voicemail notification through e-mail
  - Message Waiting Indication (MWI)
  - Support for external voicemail system
- Fax
- Instant Messaging support (softphone)
- Video support (softphone)

Management & Configuration

- Installation setup wizards
- Configurable support for WAN interface
- Configuration UI for external FXO gateway
- Route Management

Accounts

- Auto Attendant
  - Prerecorded standard destinations
  - Day/night mode, holidays
  - Dial by name
  - Anonymous call intercept
  - Black and white list management
  - Camp on
- Extensions
- Paging Groups
- Hunt Groups
  - Day/night mode, holidays
  - Distinctive ringing
  - Custom IVR recordings and routing decisions (IVR Node)
- Agent Groups
- Calling Card
- IVR Nodes
- Service Flags

Address Book

- Personal/ domain level address book
- Address book import

Trunking

- ENUM support
- Microsoft Exchange Support
- PSTN Connectivity
- CO-line emulation
- ANI number presentation
- T.38 Fax Relay Support
- Dial Plans

MoH, Paging and Intercom

- Multiple MoH sources (RTP, File, audio input)
- Multiple audio paging output (audio output, RTP multicast)

Recording and Monitoring

- Downloadable Call Detail Record
- Call Logging
- System Log Information
- Remote Log Capabilities
- CPU & Memory Usage
- DHCP & DHCPv6 Usage
- VoIP Performance Monitoring

SolidWall™ ENHANCES SECURITY WHILE ENABLING ORGANIZATIONS TO QUICKLY AND SAFELY BRING THEIR NETWORKS ONTO IPV6.

Dual-Stack Firewall and IPv6 Tunnel Endpoint

SolidWall™ is an easy-to-use network security and IPv6 tunnel endpoint appliance. It combines the functions of a dual-stack, stateful perimeter and internal network firewall, Quality of Service (QoS) capabilities, VPN endpoint, and IPv6 tunnel connectivity. The appliance is designed to enhance network security while allowing organizations to set up IPv6 connectivity even over IPv4 service provider connections, quickly and safely bringing IPv6 into their networks.

SolidWall™ provides a high level of security to networks behind it while normalizing and conditioning network traffic. The appliance also provides bandwidth management and control, redundant high-availability failover capability, and full support for IPv6 addressing, IPv6 tunneling, packet filtering and prioritization.

In addition to its advanced yet easily-managed firewall capabilities, SolidWall™ also has support for 6in4 IPv6 tunneling as a client or server endpoint. The appliance connects to existing 6in4 Tunnel Servers and can itself act as a 6in4 Server. Setup and management of tunnels is also done through wizards and the product's graphical interface.

Despite its many features, SolidWall™ is very easy to deploy and manage. A simplified installation and configuration process makes it especially suitable for organizations that lack IT security staff. Administrators can configure and manage the appliance using a secure, web-based graphical user interface, with intuitive wizards that greatly simplify configuration of many functions. A secure, restricted command shell as well as a serial communications interface are also available.

Features

IPv4/IPv6 Support
SolidWall™ provides dual-stack addressing, static routing and other basic security services over IPv6 and/or IPv4.

Dual-Stack IPv4/IPv6 Firewall
SolidWall™ allows strict control of network traffic through stateful packet filtering in IPv4 and IPv6. It has all the typical IPv4 firewall features including several types of NAT, as well as full IPv6 stateful inspection packet filtering for IPv6 and client and server support for 6in4 tunneling (IPv6 tunneled over IPv4). Rules that govern the data packet flow can be set based on previously determined factors such as traffic source, destination, or type of service provided. Sophisticated firewall rule configuration is simplified using a graphical management interface.

IPv6 Tunnel Support
SolidWall™ supports 6in4 tunneling. The appliance features a client tunnel service allowing connectivity to existing Tunnel Servers or commercial IPv6 providers. It can also act as a 6in4 Tunnel Server. Tunnels provide an IPv6 encapsulated connection over an existing IPv4 infrastructure. Such tunnels are managed through the SolidWall™ WebAdmin user interface. Depending on the tunnel server, SolidWall™ may route an arbitrary IPv6 network once connected.

Static Routing
SolidWall™ is a multifunctional appliance with support for IPv4 and IPv6 static routing. In static routing, the route that data packets should take to their destination is specified. These routes are stored in a static routing table.

IPv6 Router Advertisement and Stateless Address AutoConfiguration
SolidWall™ provides a standard, automated method for configuring IPv6 addresses for network hosts and devices in the network known as the Stateless Address Autoconfiguration. With SolidWall™'s Router Advertisement (RA) service for IPv6, the appliance can advertise the organization's delegated subnet-prefixes, thus allowing IPv6 nodes to form their unique IPv6 addresses and obtain the default route for local or global routing.

Backup and Restore
With the SolidWall™ System Maintenance utility, configurations can be easily backed up, downloaded and set aside for future use. Restoring the system is also convenient through easy uploading of backup configuration files to the appliance.

WAN Interface IPv4 Access Methods
SolidWall™ has the following IPv4 address configuration methods: Static IPv4 addressing, DHCP client, PPPoE client.

System Configuration
SolidWall™ is easy to configure and features a wide range of system functions. These features include DNS Settings, Forwarder, Interface Assignments, Certificate Manager and Setting to Factory Defaults.

Index