B.ENG. IN ELECTRONIC ENGINEERING

PROJECT REPORT

Investigation of Internet Protocol version 6

Orla McGann

97616354

Acknowledgements

I would like to thank Mike Norris, Dave Wilson, Ann Harding and everyone at HEAnet Ltd for giving me the opportunity to study a relatively new area of Networking; Martin Collier for undertaking the supervision of this project and guiding me in the direction it should take; Mark McLaughlin for listening to my panic attacks and helping me to see my potential; Conor Maguire for the never ending hardware support; James Raftery for being a FreeBSD guru; Tiarnan DeBurca for being a Windows2000 guru, Colin Whittaker for also being a FreeBSD guru; Mike McHugh for being a constant source of distraction, and occasionally information; James Healy for being a networking guru, and trusting me not to do anything mischievous with my new found power; and Ann Byrne for reminding me of her project disasters and making me feel better. Also to Robert (Bob) Fink <fink@es.net>, and Francois Baligant <francois@be.wanadoo.com> for there swift responses and guidance with my tunnelling problems.

Declaration

I hereby declare that, except where otherwise indicated, this document is entirely my own work and has not been submitted in whole or in part to any other university.

Signed: ..... Date: .....

Abstract

This project aims to provide proof of the necessity of a new Internet Protocol, (which is called Internet Protocol version six, (IPv6)). An in-depth analysis detailing the changes that are to be made to it, and its dependent protocols for the Internet will be discussed in this report. It will also contain details of a practical implementation of this new protocol developed on two different platforms. The choice of platforms to implement the new version of Internet Protocol shall be Berkeley UNIX (FreeBSD) and Microsoft Windows 2000 Professional. Sun Solaris version 8 and Linux Debian also have developed an IPv6 stack implementation, and compatible software to go with it. The platforms were chosen because they are stark contrasts to each other; one is an Open Source Operating System, and the other is a commercially developed Operating System.

The implementation of the protocol, will be described in terms of the establishment of IPv6 over IPv4 tunnels for each of the platforms. These tunnels will then be put to practical use to test the robustness, and usability of Internet Protocol version six.

The report will also include a section which contains an analysis of the IP dependent protocols, such as Domain Name Service (DNS) and the various standard Routing Protocols, and applications that have been developed for IPv6, such as Telnet, Secure Shell (SSH), and Web Browsers. It will also detail how readily available these applications and protocols are for each platform. The project will conclude with a summary of the work achieved, and suggestions for areas of study related to this field for the future.

Table Of Contents

Table of Figures

Chapter 1

Introduction

"The beauty of the Internet and the IETF lies in a simple idea - that the only standards under consideration are those already in use on the Net. Ready, fire, aim!" -- Robert X. Cringely

The growth of the Internet has been a phenomenon, which has been receiving a lot of analysis and discussion these past few years. Few who have been working with it from it's inception as ARPANET predicted this expansion, especially as the concept of a packet switched network was created by the military so that their data could not totally be destroyed in the event of another world war. Since the popularity of the World Wide Web started increasing the discussions have tended towards it being either a major achievement, or the purveyor of doom. Some scepticism exists that the exhaustion of Internet Protocol addresses is nothing more than scare mongering and that the development of a new Internet Protocol is an unnecessary, and overly time consuming endeavour, but the majority believe it is necessary, and work began on a new Internet Protocol in 1994.

While working for HEAnet on my INTRA placement, I was introduced to many new standards and technologies (such as Multicasting and Asynchronous Transfer Mode); there were also some discussions about the new Internet Protocol IPv6 because of their connection to Internet 2, the next generation Internet. Having a reasonable knowledge of the current protocol IPv4 and TCP/IP in general, this piqued my interest more than any of the others. Preliminary research brought home the interest of a further, thorough study in the area, as an all round education of Networking, and the direction it's taking in the future.

Internet Protocol version six (IPv6), is the proposed new standard from the Internet Engineering Task Force (IETF). As the opening quote implies, there is already a large interest in this new protocol, and a lot of work has been done to make it practically viable, like updating existing applications that are Internet Protocol based. This report endeavours to be an all round guide to the new version of the protocol, and to the practical implementation of it.

Chapter two of this report describes the current Internet Protocol version four (IPv4), and its failings. Then the changes that have been made which have brought about the new protocol, Internet Protocol version six (IPv6), and of how it compares to the current version, shall be described in detail. The practical implications of establishing a connection to the new IPv6 Internet called the 6Bone (a testing ground for the new IPv6 protocol.), and the various different migration strategies available, shall then be described, as will the work undertook to complete this connection for the project.

Chapter three will detail how this new protocol was put to use, by setting up two machines with two very different Operating Systems on them. It will also discuss the benefits and drawbacks for each, as a general guideline for which is the most efficient and easily established platform. Then in Chapter four will demonstrate the robustness of the set-up by testing the IPv6 compatible applications available for each platform, and will detail the differences between the two, and the problems encountered in setting them up.

Chapter five will cover an area of IPv6 that isn't possible to test due to practical reasons (not having a dedicated subnet), but are very important to the transition to IPv6 - Routing Protocols and Domain Name Service (DNS). This area of the report will be mostly theoretical. It is merely a starting point for these topics, as they are extensive areas of networking in their own right.

Finally the conclusion will summarise the work completed, what would like to be achieved in the future in this area of study, and an overall analysis of the new Protocol.

Chapter 2

Evolution of the Internet Protocol

The phenomenal growth of the Internet since its inception in the 1960's as ARPANET (The Advanced Research Projects Agency's Network), mostly due to the creation of the World Wide Web in 1990, has caused immense problems to those who regulate and maintain the Internet. From its beginnings as four nodes at Stanford Research Institute, the University of Utah, and the Universities of California at Santa Barbara and Los Angeles, it has grown to well over 15 Million nodes[1] (a node is a connection point on a network) [1].

Increased routing tables and the vast consumption of the available address space provided under the current Internet Protocol, version four, led the Internet Engineering Task Force (IETF) to conclude that a re-working of the protocol was needed.

> *…Though it is based on much-needed enhancements to IPv4 standards, IPv6 should be viewed as a broad retooling project that will ultimately provide a much-needed evolutionary re-architecturing of today's overstressed internetworks* [2].

2.1 Internet Protocol version Four

In 1973 development began on what later became known as the Transmission Control Protocol/Internet Protocol (TCP/IP) protocol in the Defence Advance Research Projects Agency (DARPA). Since RFC 791 (DARPA Internet Program Protocol specification) in 1981 there have been no changes made to the standard for the network layer of the TCP/IP protocol, Internet Protocol (IP). By January 1st 1983, everyone who was connected to ARPANET had to connect using TCP/IP, and from this point on TCP/IP

[1] At the moment, IPv4 can support approximately 4.3 billion devices (PC's, Printers, Servers etc.) with its addressing scheme. It's impossible to count the number of nodes in use for the possible 4.3 Billion possible nodes in total. Current counts work by querying the domain name system for the name assigned to every possible IP address. See http://www.heanet.ie/Heanet/hoststat.html and http://www.ripe.net/ripencc/pubservices/stats/hostcount/2001/02/ie/index.html for the Irish (.ie) domains hostcount.

became the main protocol, and its predecessor Network Control Protocol (NCP) became obsolete.

Until the inception of the World Wide Web in 1990 by Tim Berners-Lee at CERN, and up to what is generally considered the time when the Internet 'exploded' in 1995, there wasn't any need for changes to be made. Since that time, the growth in the number of domains, and the need for Internet addresses has been increasing exponentially, to the point that if the demand keeps increasing at the current rate there will be no more address space left in five years time [3]. (Appendix A has a table, which covers the rate of growth of the Internet from 1993 – 1997)

The problem of address space exhaustion has been known since the early 1990's, and measures such as Classless Internet Domain Routing (CIDR) and Network Address Translators (NAT) have been introduced to slow down the exhaustion of Internet addresses, but these are only stop-gap measures.

The Internet Network Information Centres (InterNIC) original policy of handing out IP addresses in Class Structure - Class A, Class B, and Class C, meant that thousands of IP addresses were allocated where they weren't needed, and thus went unused. CIDR was introduced in RFC1517 (Applicability Statement for the Implementation of Classless Inter-Domain Routing (CIDR)) in September 1993 as a solution to this, but the shortage of address space still lurked on the horizon. Network Address Translators were brought in to allow for networks to be connected to the Internet, but without all of their machines being allocated a globally unique IP address of its own. This is achieved by converting private internal addresses to a smaller group of globally unique addresses. Unfortunately, NAT does not support standards-based network layer security or the correct mapping of all higher layer protocols, and difficulties can arise when two organisations that use private addresses try connecting to each other [4].

IPv4 survived the rapid changes in technology for so long because of its robustness, its ease of implementation, and its interoperability. Unfortunately, the current IP lacks features needed in today's rapidly expanding Internet – security, and scalability of networks. Now quality of service (QoS) levels are required of IP. Three main things cause the scalability problems of IP: Host Configuration, Address allocation and expanding backbone routing tables.

With the majority of Class A and Class B having been allocated, routing tables were expanding 1.5 times faster than RAM capacity was developing [3]. Address aggregation was the name given to this problem.

…[Address Aggregation] occurs when various branch subnetworks, all attach to the first network in their numeric sequence, the backbone routing table only needs the network address of the first network – all the rest can be reached from it. [3]

CIDR is the current solution to the problem of Address Aggregation. Class-based address allocation means that addresses are assigned in sequential order of application, to the Internet Registry in question, and they don't have to be aggregated to a common connection path, to the Internet backbone, which helps to stop the expansion of these backbone routing tables to uncontrollable sizes.

IPv4 supports a number of fields in its packet header, such as Time To Live (TTL), Source address, Destination Address, and Identification. The address space is 32-bits long and is written as four eight bit blocks that are separated by a decimal point, e.g. 136.206.15.5. Each number can vary between 0 and 255 (256 Bits, 2^8), but 0 and 255 are reserved, and are not normally used as machine addresses. Addresses that end in .0 are called Network Addresses and, addresses that end in .255 are known as Broadcast Addresses. So if someone sends a message to 136.206.255.255 for example, everyone on the 136.206.0.0/16 network will receive the message, and will respond if necessary.

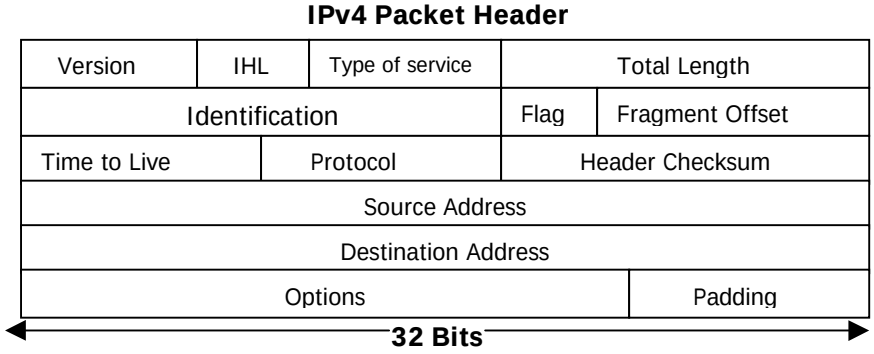


Figure2.1 The IPv4 Packet Header [5]

The security issues that frequently arise with IPv4 such as Denial of Service (DoS) attacks and other security bugs are caused by the way the above headers are misused. The fragmentation and overlapping of packets are the main causes of problems such as buffer overflow and bugs with the IP stack [6].

The IPv4 headers are of variable length in the above packet header. This in itself is a problem because the header has to tell the system its size, which is bandwidth consuming, and more difficult for routers to interpret. There are 14 fields in the packet header, some of which rarely get used, such as the Header Checksum, which is a computationally expensive way to compute errors in the packet, and the Options field, which was meant to supply information on Security and Source routing among other things, but rarely does. These have all been made obsolete, or have been merged to create different fields in the new packet header of IPv6.

2.2 Internet Protocol version Six

On November 17th 1994, the Internet Engineering Task Force approved the Next – Generation Internet Protocol, IPv6[2] as a proposed standard. The steering group developing this new standard originally proposed that the new addressing structure would be 6-Bytes (48 Bits) in length, which would provide for 65,000 times as many addresses as IPv4 has [7]. But the predicted ubiquitous nature of the Internet, and Internet connected devices (printers, scanners as well as PC's and Servers), convinced them to opt for a far greater number of addresses. It was decided that the new standard would be 128 Bits in length (16-Bytes), which is approximately 3.4×10^{38} addresses. It is difficult to imagine that number of addresses ever being used, but no matter the rate at which the Internet continues to expand, we certainly will have sufficient addresses for at least the next decade.

It is important to note that although IPv6 seems like a completely new protocol, its differences are well planned and are really a reworking of the current protocol. What worked well in IPv4 has been returned, and what did not has been changed in version six. The new addressing structure is only one of the changes that has been made. These changes are most easily demonstrated when you look at the new packet header for version six. (Appendix B contains the full IPv6 header taken from RFC 1752).

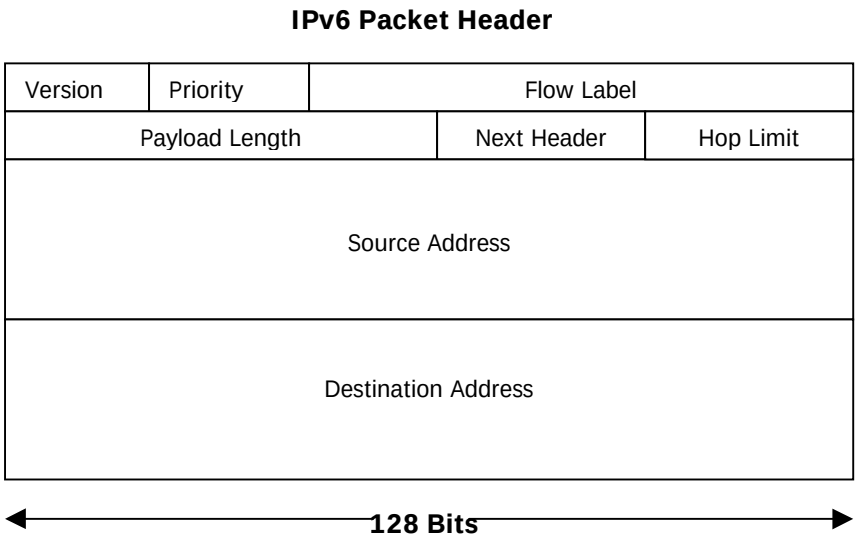


Figure 2.2 The IPv6 Packet Header [5]

[2] This is sometimes referred to as IPng - IP Next Generation

As with IPv4, the Packet Header has a version, source and destination address fields. It is easily noted though, that the new header is considerably more simplified than its predecessor. It has only 8 fields compared to the 14 fields that IPv4 has. This obviously makes routing more efficient. The reduced number of fields in the new header has been achieved by amalgamating some of the previous fields into one field for IPv6. For example, the functionality of the Type of Service field in version four has been moved to the priority field and the flow control field. These are used to provide support for real-time delivery of data (so called Quality of Service).

The Header Length field has been abolished altogether as IPv6 has a fixed header length for all packets. The payload length field has replaced the Total Length field. It is assumed that the length of the header is always 40 bits, so this doesn't actually hold any value for the length of the header. The Payload length field can register a payload up to 64,000 bits in length, and even more (in the form of *'jumbograms'*) if its value is set to zero, and the optional extension header is added. This is further explained later on in this chapter

The Time To Live (TTL) field has also been changed for this version of the protocol, to the Hop-Limit field. Although the name has been changed the field still does the same thing – It's used by routers to decrement a maximum hop value by one, for each hop in an end-to-end route. When the hop-limit value is decremented to zero the packet is discarded. The new version of this field can hold a value up to 255 hops (8 bits); this is hopefully enough for even the largest imaginable networks of the future.

Although the 16-byte IPv6 addresses are four times the size of the 4-byte IPv4 addresses, the restructuring of the packet header means the new protocols header is only twice the size of the current one [2]. This will offset the bandwidth cost of having larger address fields (128 Bits instead of 32 Bits) that need to be transferred with each packet. IPv6 has been designed with a view to keeping header overhead to a minimum. This was achieved by moving non-essential and optional fields from version 4, to extension headers in version 6. It's important to note, that IPv4 and IPv6 headers are not interoperable. A host or router must run both implementations of the protocol in order to recognise and process header information, if both protocols co-exist on a network. (Appendix C contains a table outlining the major differences between IPv4 and IPv6).

In IPv4, an options field was added to the packet header, to allow for optional information relevant to the routing process such as security and source routing. This has been replaced in version six by flexible extension headers that come after the primary packet header and before the transport header and payload data (As can be seen in Figure 2.3 below.) Unlike the options field in the IPv4 header, which only supports 40Bytes of options, the size of the new extension headers are constrained only by the size of the IPv6 packet.

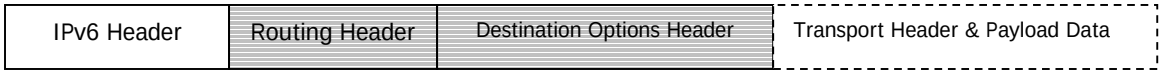


Figure 2.3 IPv6 Extension Headers [5]

These optional extension headers provide the means for supporting much-needed features like security, fragmentation, source routing and network management capability. They are also much easier to implement in version six, than in version 4 [5]. These extension headers also impact on the old Protocol Type field of version four, which is used to indicate whether TCP or User Datagram Protocol (UDP) is used within the datagrams payload. Now called the Next Header field.

The proposed order for the extension headers for the new standard are as follows:

- (Primary IPv6 header)
- Hop-by-Hop options header
- Destination Options header –1
- Source Routing header
- Fragmentation header
- Authentication header
- IPv6 Encryption header
- Destination Options header –2
- (Upper-layer headers)
- (Payload)

Each extension only occurs once except for the Destination Options header, which has two instances. The first is for carrying information to the destination listed in the IPv6

destination address field, and the second is for optional information that is only to be read by the final destination.

The Fragmentation header has been updated in version six, so en-route routers can no longer fragment the packet as it passes; only the source nodes are allowed to do this. If a packet is received that is too big, it is discarded, and an Internet Control Message Protocol (ICMP) message is sent back to the originator to inform it that the packet was dropped. This feature fixes the problems associated with fragmentation in IPv4, such as the overlapping of packets and stack bugs.

Aside from the size difference in the address fields of the old and new protocols header, there is a notable difference in the abilities of each to provide an advanced hierarchal address space that allows for efficient routing architectures. As has been previously mentioned in this chapter, IPv4 was designed with a class based allocation system in mind (see Figure 2.4). This does not, however, cater for a hierarchy that would allow for a single high-level address to represent many lower-level addresses – much like the way our telephone system for country codes and area codes work; this method allows for long haul calls to be patched efficiently to their destination using only a portion of the full phone number. A similar implementation for IP would obviously be the most efficient way of keeping routing tables manageable and efficient,

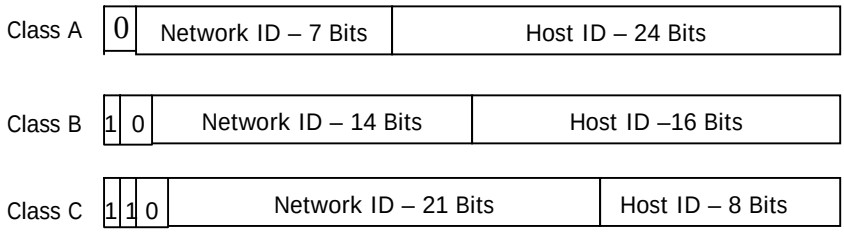


Figure 2.4 IPv4 Class based system for address allocations [5]

When the IETF were designing IPv6 they decided to build on the experiences gained from implementing IPv4, and began to structure the allocation of the new addressing architecture better. The idea of how addressing in the telephone network works provided the most simplified and accountable addressing scheme that could be partitioned into a flexible and globally usable routing hierarchy. Top Level Aggregators (TLA) would be at the top of

this hierarchy – mainly international registries like the Higher Education Authority Network (HEAnet) and the Irish Domain Registry (IEDR) in Ireland. These would act as 'exchanges' for worldwide traffic coming to Ireland, where peering could be established[3]. These Top Level Aggregators would then allocate blocks of addresses to Next Level Aggregators (NLA), which would comprise of the larger Internet Service Providers (ISP) and corporate giants' networks such as Intel, Sun Microsystems and Cisco. In the cases where an NLA is a provider of Internet access (e.g. ESATnet), it in turn allocates addresses to its subscribers. Routing tables are kept efficient as NLAs with a common TLA will have the same TLA prefix at the start of their addresses, e.g. 3ffe:b00:c18:1fff:0:0:0:193, where 3ffe would be the TLAs prefix.

IPv6 will therefore be allocated according to geographical location and provider location (Network Topology). Address allocation by provider divides the hierarchy along the lines of large ISP's and therefore is not necessarily dependent on location. Geographic allocation divides the hierarchy strictly on the basis of the location of the Provider and its subscribers (in the same way as the telephone network operates). These have their drawbacks – large networks don't always conform to geographical boundaries, and some large networks may have more than one ISP, but it is still the most efficient way to allocate addresses.

Aggregation based allocation is built on the idea that a number of 'exchanges' exist, where long haul providers/telecom companies interconnect (as in Figure 2.5), and the new IPv6 address hierarchy has been designed with this structure in mind. It is hoped that the rigorous structure of IPv6 allocation will go a long way to keeping the backbone routing tables of the Internet manageable, no matter the rate at which the Internet continues to grow. (Appendix D contains a table of the breakdown of the allocation of IPv6 address space).

IPv6 embodies a number of addressing systems [8] such as

- Unicast,
- Anycast,
- Multicast.

Unicast is the current method of routing, where a packet that is addressed to one machine/host is sent and only one machine/host receives it (usually the machine named as

[3] As is currently the case, but on a much more structured basis now instead of using any ISP that has the connections.

the destination of the packet). RFC 2373[4] allows for the accommodation of load-balancing systems, where multiple interfaces may use the same single address, as long as these interfaces appear as one interface to the IPv6 implementation on the machine.

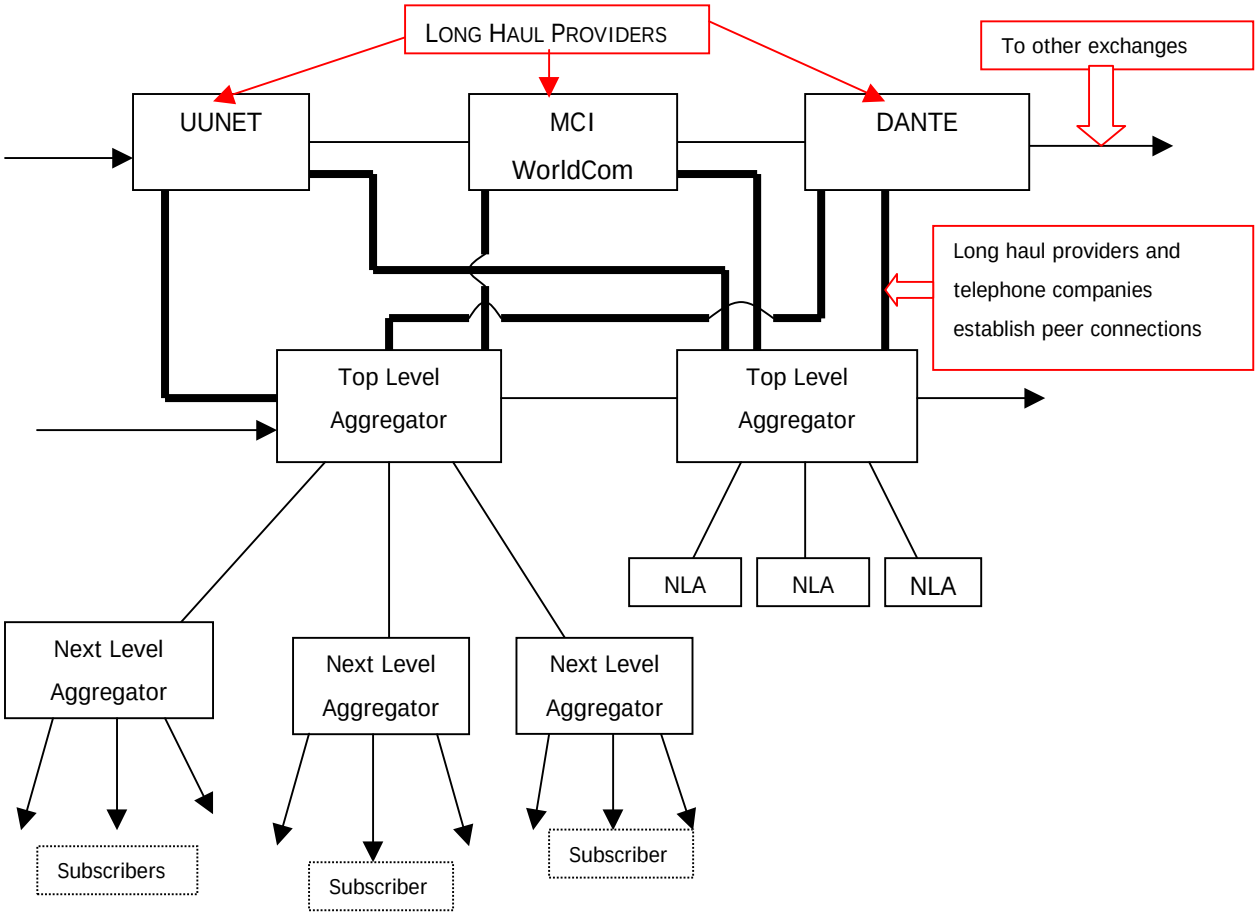


Figure 2.5 Graphical View of the Aggregated Allocation Structure

IPv6 unicast addresses can be of the following types:

- Aggregatable global unicast addresses,
- Link-local addresses,
- Site-local addresses,
- Special addresses,
- Network Service Access Point (NSAP) and Internetwork Packet Exchange (IPX) addresses.

[4] IP version 6 Addressing Architecture

Anycast addresses identify multiple interfaces (typically belonging to a number of different nodes). A packet sent to an anycast address is delivered to one of the interfaces identified by that address (the "nearest" one, defined by the routing distance). Anycast addresses are used for ‘one-to-one-of-many’ communication, with delivery to only a single interface. Anycast addresses are particularly useful in the situation of an organisation that has many connections to its ISP, and all connections have the same anycast address. In this case, the organisation could have redundant access points in case that any of its normal access points goes down or is not responding for some reason. It is also being used to automate mirroring systems for websites that are frequently visited by people for downloads – such as http://www.FreeBSD.org, and http://www.linux.org, where the data for downloading is stored on one server, e.g., in Berkeley, California (in the case of FreeBSD) and that data is replicated on a number of different websites all over the world. Anycast addressing would eradicate the need to go to the home site and find the nearest working mirror site to where you are, as it would automatically redirect to the “nearest” mirror.

Multicast addresses identify multiple interfaces (as the name implies). A packet sent to a multicast address is delivered to all interfaces identified by that address. IPv6 multicasting operates in the same way as it does for IPv4. Arbitrarily located nodes can listen for multicast traffic on an arbitrary IPv6 multicast address. IPv6 nodes can listen for multiple multicast addresses at once. A multicast address is used for ‘one-to-many’ communications, delivering obviously, to multiple interfaces. The IPv4 broadcast address (i.e. the 255 suffix) has been removed from IPv6 and its functionality has been replaced by multicast addressing. Multicast addresses are generally used for video conferencing and the like.

At first glance the addressing syntax of IPv6 seems completely unintelligible compared with the current 32 bit eight-by-four addressing. For IPv6 the 128 bits have been broken up into 16 bit boundaries and each boundary is then separated by a colon (as opposed to a dot in IPv4). Then each 16-bit block is converted to a 4-digit hexadecimal number, to give the address in its current state. For example, the IPv6 address 3ffe:b00:c18:1fff:0:0:0:408 is derived from the binary sequence

0011111111111110: 0000101100000000: 0000110000011000: 0001111111111111
0000000000000000: 0000000000000000: 0000000000000000: 0000010000001000

The above IPv6 address has been simplified by removing the leading zeros where they occur within each block. Each block must have at least one digit, except where there is a continuous set of zero 16 bit blocks, in which case the colon hexadecimal format (as the IPv6 address format is known as) can be compressed to "::" . So the address would be further minimized to 3ffe:b00:c18:1fff::408.

Obviously remembering these addresses is out of the question, no more than with IPv4, and Domain Name Servers (DNS) will be discussed in Chapter 5 as a means to providing intelligible names to these strings of numbers and characters.

Probably the most useful feature of the new IP is its Autoconfiguration of addresses. IPv6 supports both stateless and stateful address configuration. The latter occurs when there is a Dynamic Host Configuration Protocol (DHCP) server is available, and the former when there is not. With stateless configuration, the host/device automatically configures itself with an IPv6 address for the link (a so called 'link-local address') and with addresses derived from the prefixes advertised by local routers. In fact there isn't even a need for there to be a local router, hosts on the same link can automatically configure themselves with link-local addresses, thus making the manual configuration of IP addresses redundant[5]. This is hugely important for large networks that have to relocate or add new hosts on a frequent basis, whose IP addresses have to be re-configured. It also takes the error out of establishing networks, where IP conflicts might occur when two machines are mistakenly given the same IP.

Autoconfiguration is achieved using the Neighbour Discovery (ND) Protocol. Under IPv4 Address Resolution Protocol (ARP) and Internet Control Message Protocol (ICMP) provided this service. ND has been refined to include these protocols for IPv6, and although its disguised under a new name it is really only a set of ICMPv6 messages that allow nodes on an IPv6 network to discover the link layer addresses, and to obtain and advertise network parameters, such as the address prefix, and reachability information.

Basically, a machine/host starts autoconfiguration by self-configuring a link-local address for temporary use. The address is usually formed by taking the network card Local Area Network (LAN) interface address. Once it forms this address, the host sends out an ND message to see if it's unique. If it's not, an ICMP message will come back to the machine complaining that it's already in use. If the machine receives no ICMP message back, it

[5] IPv4 also uses dynamic host configuration, but it's by no means standard as it is in IPv6

knows the address is unique and keeps it. If a message comes back saying that the address is already in use, then a different token is used, like an administrative token or even a randomly generated token.

Using its link-local address, the machine then sends out an ND router solicitation, using a multicast address[6]. Routers respond to this solicitation message, with a unicast router advertisement that contains the network prefix(es) and also indicates a valid range of addresses available on the subnet. The Neighbour Discovery protocol is demonstrated as follows:

[6] IPv6 has defined several permanent multicast groups for finding resources on the local node or link, including an DHCP server group, an all-hosts group (which sends a message to all hosts) and an all-router group (which sends a message to all routers).

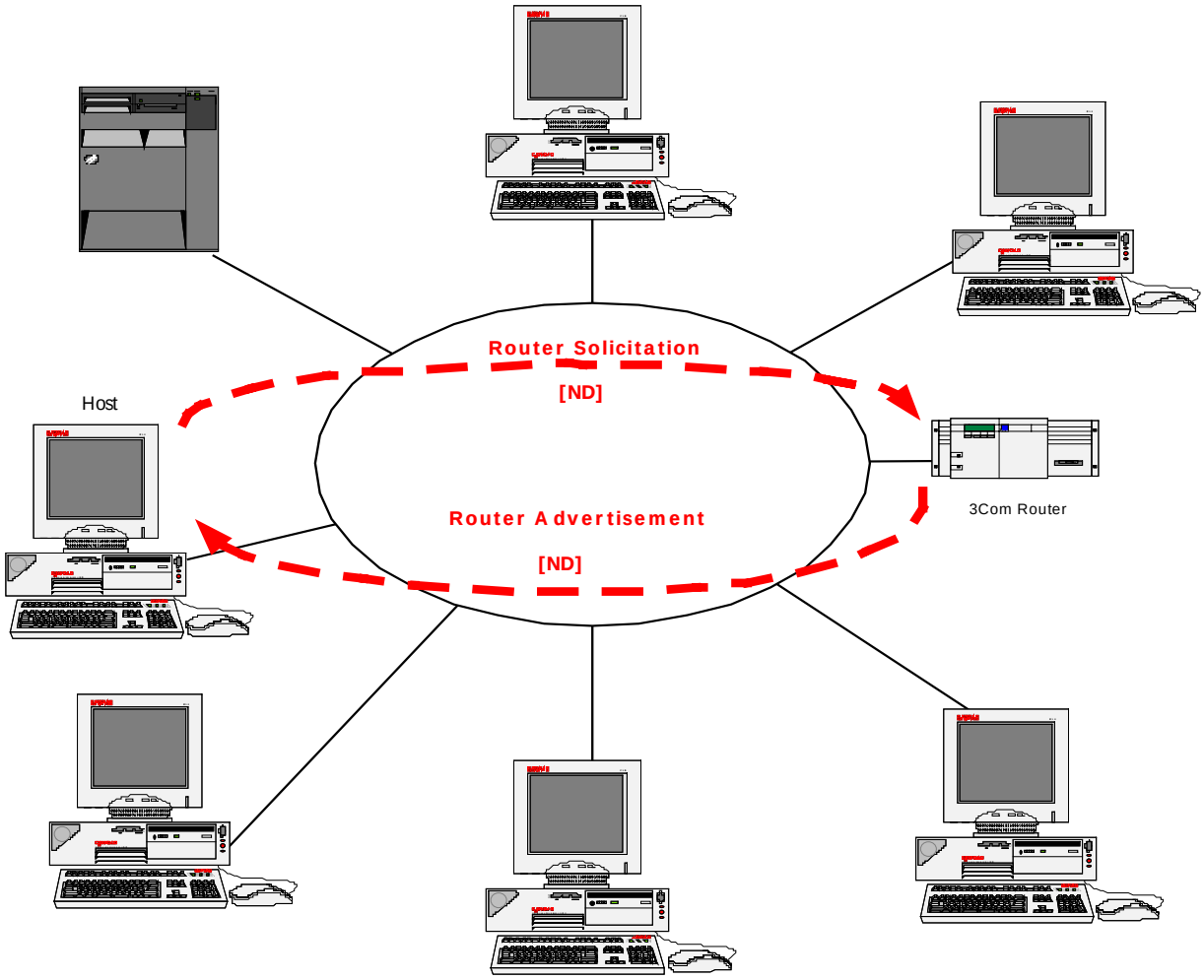


Figure 2.6 Neighbour Discovery Message Exchange [5]

Previously IP addresses were assigned to whoever wanted them by the various Top Level Internet Registries; such as RIPE-NCC (in Europe), APNIC (in Asia and the Pacific) and ARIN (in Northern America), and they were obtained independently of the ISP they intended to use. So it was possible for users to change ISP without having to go and change their IP address. But with IPv6, when an organisation wants to change ISP, it has to change its address (because of the way addresses are assigned.) This is only made an acceptable address assignment model because of the ease with which whole networks can be re-numbered using IPv6s autoconfiguration.

IPv6 aims to establish three important security services in the form of:

- Packet Authentication,
- Packet Integrity,

- Packet confidentiality

All of these security features are encapsulated in one of IPv6s optional extension headers – the Authentication header, as previously noted in this section. It provides integrity validation, guaranteeing network applications that the packet did actually come from where it claims to have come from. This will combat the increasingly frequent attacks on networks, where the cracker configures a host to impersonate another, to gain access to private resources. Under IPv6s Authentication headers, hosts establish a standard based security association that is based on the exchange of the keyed MD5 digest algorithm (but those implementing the protocol can use a different algorithm, as long as it's appropriate.)

Before each packet is sent, the header creates a 'checksum' based on the key supplied with the packet. This checksum is re-run on the receivers end and compared to the original. In this way authentication is proved, and the sender of the packet can guarantee that it hasn't been interfered with en-route.

The Authentication header provides security in the form of eliminating 'host spoofing' and packet modification cracks, but it doesn't safeguard against packet sniffing and snooping as the packet is transferred over the Internet (and other large networks). The IPv6 Encryption header aided by the Encapsulating Security Payload (ESP) header is used to do this. They are also part of the optional extension headers that I discussed previously. ESP encryption methods offer high levels of privacy and integrity for packets, which you would rarely find available in an IPv4 environment, except in the case of certain secure applications like Secured Shell (SSH), and secure Web Servers.

Most importantly, ESP provides encryption on the network layer, so it is standardised and all applications can avail of it. IPv6 ESP is also used to encrypt the transport-layer header and payload (i.e. either TCP or UDP) that follow the optional extension headers (see Figure 2.3). It can also be used to encrypt the entire IP datagram. Obviously fully encrypted datagrams are more secure than transport layer encryption only, due to the fact that the headers of the datagram are also encrypted and not available for traffic analysis like packet sniffing. ESP uses the Data Encryption Standard – Cipher Block Chaining (DES-CBC) as the default, and will be required for all implementations.

2.3 Migration strategies from IPv4 to IPv6

The current spate of hype about the exhaustion of IP(v4) addresses has provoked debate about the urgency of a transition from IPv4 to IPv6. It is obvious that the changes IPv6 will bring about will improve greatly on the current set-up, and is necessary in both the short and the long term for network dependent industries. How to go about this transition, however, is not quite as clear-cut. There is a feeling among some that until there is complete address space exhaustion; the current set up will suffice. Others on the other end of the scale are lobbying for a complete upheaval and transferral to IPv6 as soon as possible. It is clear that IPv4 and IPv6 will have to co-exist until the changeover is completed. Also many organizations are completely dependent on the Internet for their daily work and they cannot therefore tolerate the loss of connectivity for the replacement of the IP protocol. Thus, there will not be a flag day in which IPv4 will be turned off and IPv6 turned on, since the two protocols can coexist without any problem.

In anticipation of this, the IETF expanded the protocol to ensure that the transition would be as smooth as possible, and Cisco and Nortel Networks have already begun producing IPv6 capable routers. It is going to be a very expensive change for companies who did not prepare by purchasing upgradeable equipment, as it means a whole new networking structure for their organisation. The rate at which IPv6 transfers will be dependent on how prepared companies are in this respect, and how quickly IPv4 will become obsolete. For companies who planned ahead for this change, it should only be a matter of upgrading their Operating System[7] (OS) and their Internetwork Operating System[8] (IOS).

Transition methods have been designed to give maximum flexibility to Network Engineers, for how and when they need to upgrade their networks. Thus, IPv6 can be deployed in routers first, or in machines/hosts first, or in a limited number of adjacent or remote hosts or routers for the purposes of testing. [5] It has also been assumed by the IETF engineers, that compatibility with IPv4 devices will be necessary for some time (while companies structure their upgrade budget), so upgraded devices will have the option of retaining their IPv4 address as well. A number of functions have been incorporated into the

[7] For their computers

[8] For their routers

IPv6 standard including dual-stack hosts and routers, translators, and tunnelling IPv6 via IPv4 to facilitate this.

2.3.1 The Dual Stack Transition Method

When a number of nodes have been converted to IPv6, they will most likely need to maintain communication with the existing IPv4 nodes (as is now the case with the establishment of the 6Bone[9]). This is accomplished by means of the dual-stack IPv4/IPv6 approach. Dual stack machines (and routers) can handle both IPv4 and IPv6 packets, with a separate stack implemented for both. When a dual-stack is being run on a host, the host will have access to both IPv4 and IPv6 services. Routers running a dual-stack will handle IPv4 and IPv6 traffic for an IPv4 node and an IPv6 node. These two protocols act independently of each other and are unable to communicate with each other.

Otherwise, the machine can be configured with an IPv6 address that is IPv4 compatible. Compatible addresses are created by adding leading zeros to the IPv4 addresses to fill it out to the full 128-bits of an IPv6 address, this is demonstrated in Figure 2.7. So they have syntax of the type "::136.206.15.5", which is not to be confused with the IPv4 syntax of 136.206.15.5. Dual-stack nodes should also be able to use DHCP to autoconfigure their IPv4 addresses, the IPv6 address can be configured as normal (as described in the previous section).

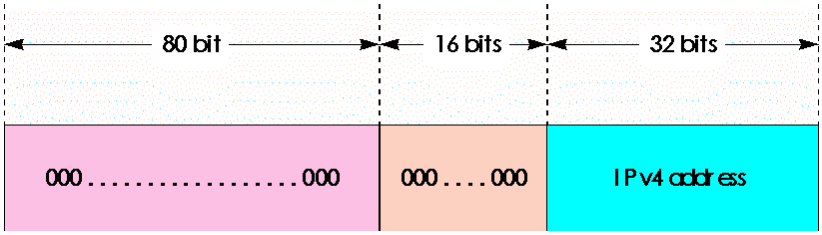


Figure 2.7: IPv6 addresses with embedded IPv4 addresses [9]

Most of today's routing equipment is already made with a view to running multiple network stack components (i.e. IPv4, IPX, NetBIOS etc.) so adding another one in the form of IPv6 is a fairly trivial adjustment.

[9] The test bed for IPv6

The migration from IPv4 to IPv6 has to incorporate the following, in order for there to be a smooth and easily implemented transition [9]:

- IPv6 and IPv4 hosts must be interoperable,
- The use of IPv6 hosts and routers must be diffused in the Internet in a simple and progressive way, with a little inter-dependence,
- Networks Engineers and final users must feel that the migration is simple to understand and will be easy to implement.

Below is a possible organisation of the stack, to give a better idea of how the dual-stack will work within the whole system from end-user to the Internet or network they're connected to.

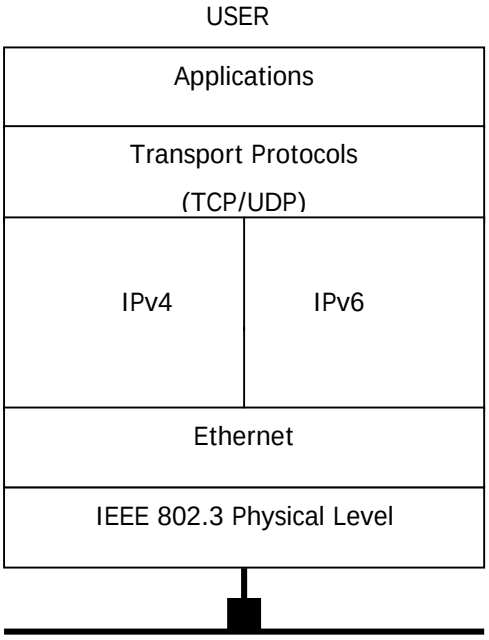


Figure 2.8 The Dual Stack Approach [9]

So, if the destination address is in IPv4 format, then the IPv4 protocol stack is used; if the destination address is in IPv6 format, then the IPv6 protocol stack is used, and if the destination address is an IPv4 address embedded in an IPv6 address, then IPv6 is encapsulated within IPv4 (for the purposes of tunnelling).

2.3.2 IPv6 over IPv4 Tunnelling

…The dual stack approach doesn't necessarily require the ability to create tunnels, while the ability to create tunnels requires the dual stack approach. In general, both approaches are provided by IPv6/IPv4 implementations. [9]

While the IPv6 routing infrastructure is being developed (as described in greater detail in Chapter 5), routing will continue in IPv4. It is necessary for IPv6 traffic to be forwarded (for the purposes of testing mainly at this time), and tunnelling provides a means to this, using the current IPv4 networks. This is demonstrated in Figure 2.9 below:

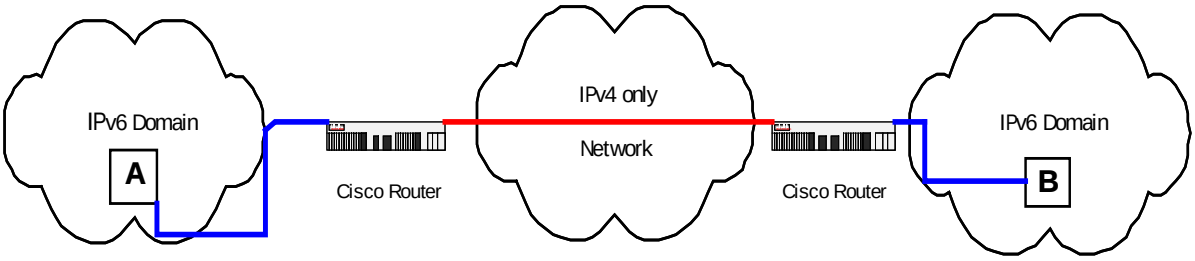


Figure 2.9 IPv6 over IPv4 tunnelling.

In this scenario, a packet is sent from host A in its native IPv6 format. The packet is then retransmitted over the current IPv4 network to its destination, as an IPv4 tunnel. When it reaches the second router, it transmits the packet to its destination in its original format. In this case the routers manage the tunnel.

Another method of tunnelling is to use Encapsulation. This is where the IPv6 packet is encapsulated inside an IPv4 packet. With this method, the packet is sent under the guise of it being an IPv4 packet, so has the address of the routers as its destination addresses. The packet is transmitted by the addition of an IPv4 header, with the IPv6 packet as the payload, complete with its own native IPv6 headers. When it is received at the other router the IPv4 header is removed to reveal the original IPv6 packet, which can then traverse the IPv6 network. Figure 2.10 shows the encapsulated and the normal IPv6 packet:

Figure 2.10 IPv6 Packet and Encapsulated IPv6 packet

To accommodate different administrative needs, there are two types of tunnelling methods: automatic and configured [5]. Automatic tunnelling uses the encapsulation method described above.

Configured tunnels are established by the administrator manually defining IPv6 to IPv4 address mappings at the tunnel endpoints. At the entry point of the tunnel, a router table entry is defined manually to decide what IPv4 address is used to traverse the tunnel. The traffic is then routed dynamically, without the knowledge that IPv6 is involved at all. The IPv6 address does not have to be compatible with the IPv4 address.

If there are a number of machines/hosts on an IPv4 network, there is a method of tunnelling IPv6 traffic between nodes on different subnets of the network known as 6to4 [10]. It is designed for connecting a number of IPv6 hosts over an existing IPv4 network. It works by creating a unique IPv6 address prefix, to give isolated IPv6 sites their own address space; it's best described as a "pseudo ISP" providing IPv6 connectivity. 6to4 can be used to communicate directly with other 6to4 sites, without the need for an IPv6 compatible router. The IPv6 traffic travels as an encapsulated packet. Figure 2.11 shows the 6to4 set up.

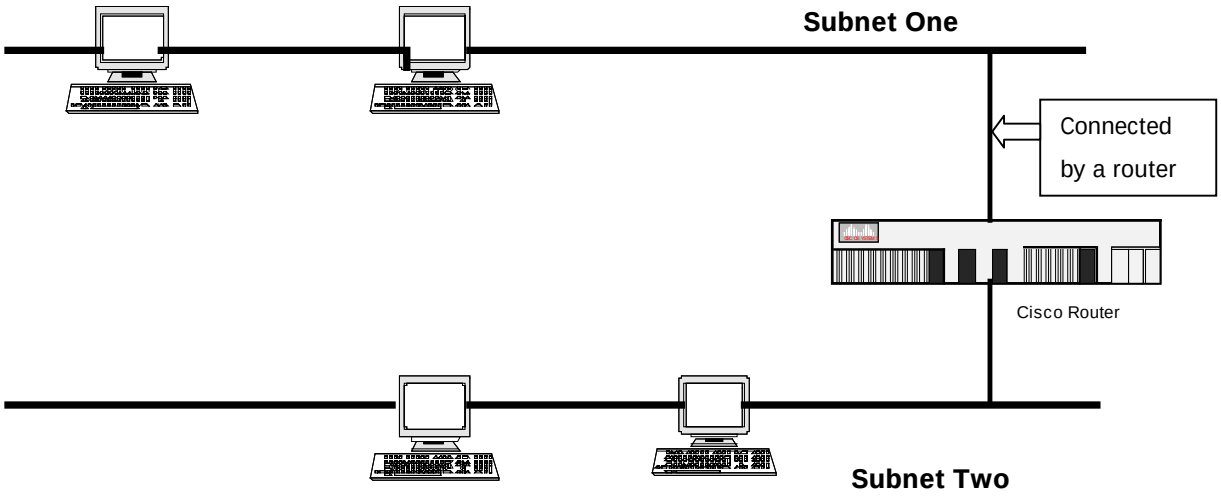


Figure2.11 Multiple nodes on separate subnets using 6to4 to communicate across an IPv4 router.

The only real requirement for the 6to4 configuration is to have a globally unique routable IPv4 address (i.e. not a private address or one generated by a NAT). This address will act as the 6to4 gateway, and must be run on a machine with the IPv6 protocol.

There are many options for Network Engineers to choose from in order to change to IPv6 for every network set up. A configured tunnel was implemented on both of the machines, for this project, and is described in the next chapter.

Chapter 3

Implementing IP version Six on Two Platforms

For the purpose of testing the new IP, it was decided that testing on two separate platforms would help give a greater understanding of how the deployment of IPv6 is progressing, and to compare two of the current implementations. FreeBSD was chosen initially, because it was the first Operating System (OS) that supplied an IPv6 stack. Windows 2000 was chosen as a contrast to the FreeBSD operating system (one is Open Source, and the other is a Commercial Operating System), and Microsoft had just released the new version of their research IPv6 stack as I was undertaking this project.

3.2 The Windows 2000 Machine

Windows 2000 machine is running on an AMD K6 500 processor with 128Mbytes of RAM. The installation of the Windows 2000 operating system posed a few problems at first. The original machine that Windows 2000 was installed on had air circulation problems, which caused the motherboard to overheat which then caused the machine to crash at random intervals. There were also a lot of problems getting a network card, which didn't cause interrupt request (irq) conflicts with the other hardware in the machine. There were also some notable differences between Windows 2000 and Windows NT4, Windows 2000's predecessor – such as the network dial-up properties being moved from the Control Panel in the Settings menu, to a panel of its own. Once a familiarity with the changes was achieved, and the problems with the hardware were ironed out, the installation of the operating system was relatively trouble-free.

The IPv6 stack for Windows does not come as standard protocol, as it is only a research project at the moment. The Microsoft research web site contains their test tcp/ipv6 stack, and IPv6 protocol, and is available to download from there[10]. Once the files were downloaded and 'unzipped' they had to be moved to the system directory in C:\WinNT\system, in order for the dynamic link library files (.dll) to work correctly.

The next stage was to add the IPv6 protocol to the list of available protocols on the machine. This was done by going into the settings menu, to the network dial-up properties,

[10] http://www.research.microsoft.com/msripv6/

to the local area network connection. Clicking on the “properties” button here brings you to a menu of the protocols on the machine. There is an install option, from which you can then ‘add’, and select the Network Protocol you want from a list, in this case IPv6. When you click ok after adding the IPv6 protocol, you can see that it has been added to the list of protocols on the machine. Unlike IPv4, you can’t manually configure the IPv6 address from the properties menu when you click on the protocol, as the IPv6 address is automatically configured, as is expected.

Once this is completed, configuring the tunnel for the machine (as described in section 3 of this chapter) can begin. There were very few problems with the Windows 2000 machine once the operating system was correctly installed, and there were no hardware conflicts. In February, Service Pack 1 had to be installed to fix some bugs in the original installation, and to add 128-bit encryption to Internet Explorer version 5.5.

Some other software for the purpose of testing the protocol installation was downloaded at a later date, such as PuTTY, Tera Term Pro, Apache Web Server, Microsoft IRC client (MIRC), and an IP bouncer. These were installed directly from the web site (as listed in chapter 4 where relevant), without any problems.

3.2 The FreeBSD Machine

The FreeBSD machine is an Intel 486 with 64 Mbytes of RAM, a considerably less powerful machine than the Windows 2000 machine. When I first installed the operating system, I used version 4.1.1 RELEASE, the then most up to date version of the OS. A couple of weeks later it was updated to version 4.2 RELEASE, but there was no need for me to upgrade at that point. Before installing the operating system I had to partition the hard-drive with a FreeBSD partition using the software ‘FDISK’.

The FreeBSD operating system was installed by booting from two floppy disks, mfsroot.flp and kern.flp, that held image files of the root file system, and the FreeBSD kernel respectively. These image files were downloaded from the FreeBSD website at http://www.FreeBSD.org/doc/en.US.ISO_8859-1/books/handbook/install-guide.html.These were then used to boot the machine into a root file system, and to load the kernel onto the machine. Once this was done, an installation menu allows you to pick the type of installation you would like –the file transfer protocol (ftp) installation was chosen, as the CD ROM on the machine didn’t work properly. From there, another menu which lists the various mirror-sites of the operating system all over the world is brought up. The Irish ftp mirror site was

chosen, which is hosted by Esat at ftp.esat.net. The download and installation of the operating system took overnight to complete. The type of Internet connection you have and the speed of your computer dictate how long this installation will take if done over FTP.

The installation guide was then used to set up user accounts on the machine, to configure the IPv4 address of the machine and it's netmask and broadcast addresses. An X server and client was then set up for ease of use of the computer – this creates a windows based operating environment to work from. The "GNOME" desktop, with the "TVM" window manager were chosen for ease of usability and low system usage. A number of packages were installed at this point for the testing section, such as Eterm, SSH, and Telnet. When these were all installed and configured correctly, the IPv6 tunnel was set up (as described in section 3 of this chapter).

FreeBSD is IPv6 compliant "out-of-the-box", so there was no real patching of the system to get the protocol working. KAME[11] is a project that is the joint effort of seven companies in Japan who are working to provide a free IPv6 and IPsec (for both IPv4 and IPv6) stack for BSD variants to the world. FreeBSD supports this software as a standard package that you can choose to download when you're choosing the packages you want on your machine at the time of Installation. The KAME project is also responsible for having made IPv6 compliant versions of most of the major software distributed with FreeBSD such as Apache, Mozilla, Secure shell (SSH), Internet Relay chat (IRC), Berkeley Internet Name Domain (BIND), Sendmail, Lynx, Point-to-Point Protocol (PPP) and Squid.

For the purpose of testing the IPv6 protocol, I also downloaded a number of software packages to test for their IPv6 compatibility. They were as follows:

- Apache Web Server,
- Mozilla Web Browser,
- IRC Server,
- BitchX IRC Client

Ping6 and traceroute6 come as standard packages with the IPv6 stack.)

By the time I went to install these packages, the version of FreeBSD I originally installed, was out of date so I had to upgrade the installation, and therefore the kernel. This is a tricky and long process if you've never done it before (FreeBSD estimate it takes five hours for a **make installworld**, the command that updates your files, on a Pentium II with 64 Mbytes of RAM). The upgrade in total took almost four days on the 486 machine.

Obviously this suggests a need for running the tests on a more powerful machine, but other than this instance, FreeBSD dealt well with the tasks it was asked to perform.

Setting up and administrating the machine was the biggest task for the FreeBSD machine, as I had no experience of installing a Unix operating system prior to this. Obviously it is necessary to have a good knowledge of how Unix works, and of some of the basic commands needed for file management. However, there are very good tutorials and man pages to aide with the installation on the web page at http://www.freebsd.org. FreeBSD is well equipped to deal with the transition from IPv4 to IPv6, and with the help of the people working on the KAME project, they also have more IPv6 compatible software than any other platform, and they are developing it at a much faster rate – a testament to Open Source products.

[11] http://www.kame.net

3.3 Establishing the IPv4 – IPv6 tunnels

The theory behind how tunnelling works (as described in Chapter 2) is very important, but more so is the actual implementation of this theory. Figure 2.11 is a diagram of how the set-up for testing the IPv6 protocol works. Both machines have a tunnel to the 6Bone that goes through the IPv4 network using configured tunnels. The next sections describe this set up in detail for two different platforms.

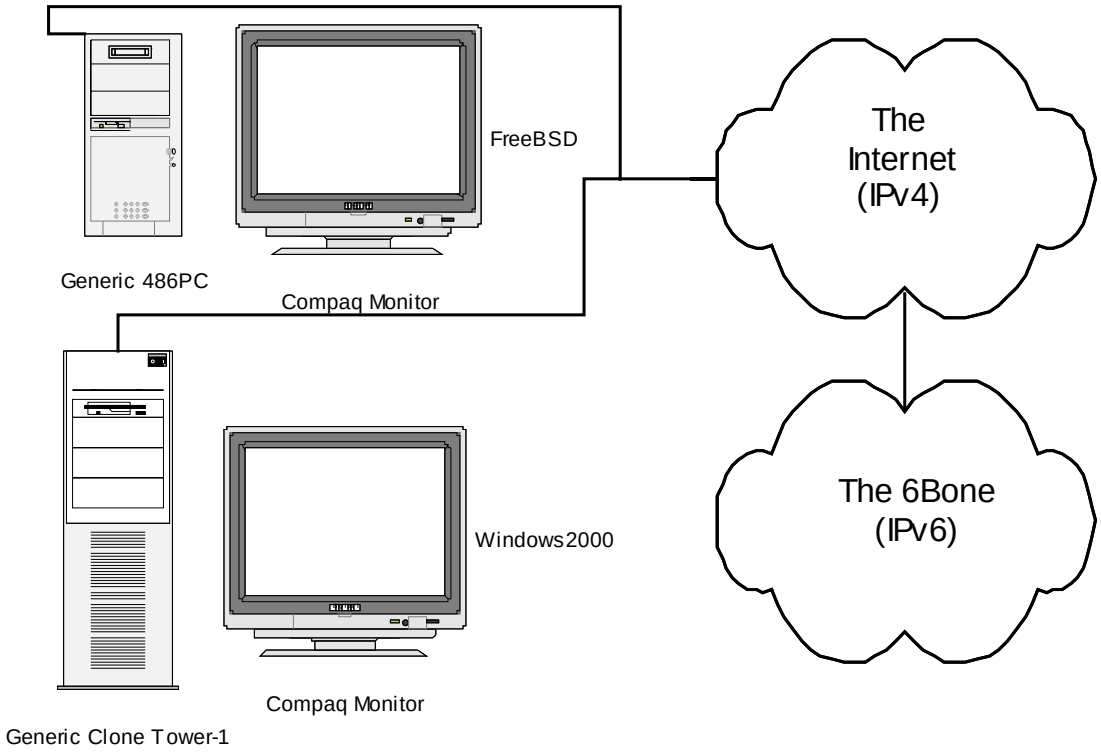


Figure3.1 The implementation of IPv6 over IPv4 Tunnelling

3.3.1 On FreeBSD

For the purpose of describing the tunnels it is assumed that the machines are correctly configured with the IPv6 protocol stack. Routing equipment capable of processing IPv6 and the routing protocols that are used to forward this IPv6 traffic are still being developed, and are not widely available for the purpose of testing the IPv6 protocol, so most people connect to the 6Bone (The IPv6 test bed) by way of an IPv6 over IPv4 tunnel as described previously. There are a number of groups that provide these tunnels available on

the Internet[12]. If you don't have routing equipment available to establish your own tunnel and route your own packets, this is the quickest and easiest way to go about connecting to the 6Bone.

All the routing table configuring that needs to be done, is done by the tunnel providers, so all that remains on the users side of the tunnel is to set up the interfaces to listen for IPv6 traffic and then to create the tunnel between the machine in question and the server the other end.

To set up the interfaces on FreeBSD you need to execute the following commands:

- `gifconfig <available interface> 136.206.25.248 206.123.31.102`
- `ifconfig <available interface> inet6 3ffe:b00:c18:1fff:0:0:0:21f 3ffe:b00:c18:1fff:0:0:0:21e prefixlen 128 alias`
- `ifconfig <available interface> up`
- `route add –inet6 default 3ffe:b00:c18:1fff:0:0:0:21f`

The first command **gifconfig**, is used to configure generic IP tunnel interfaces, (i.e. gif0), using the physical address. This command takes a number of parameters in the form **gifconfig [interface] [af] [physsrc physdest]**. The parameter *interface* is a 'string', i.e. a group of numbers and characters, in the form of 'en0'. The parameter *af* specifies the type of protocol that is to be used, i.e. inet or inet6, depending on whether it's IPv4 or IPv6. *Af* is has the default parameter of inet. *Physsrc*, and *physdest* are the parameters, which describe the source address and the destination address respectively. **Gifconfig** is used to create IPv6 over IPv4 tunnels – using it to try and create IPv4 over IPv4 tunnels could create an infinite routing-loop.

The next command **ifconfig** is used to configure interface parameters. The command takes the form: **ifconfig [interface] [address family] [address] [dest address]**. The *interface* parameter is the same as for **gifconfig**, and the *address family* parameter is the same as the *af* parameter in **gifconfig**, and also supports the "atalk", "ether", and "ipx" protocols. The *address* parameter obviously contains the address of the host, and the *dest address* allows you to specify the address of a correspondent on the other end of a point-to-

[12] In line with IPv6s address allocation policy, you should connect to the nearest provider of these tunnels to you, but at the time of testing there were no Irish providers of tunnels, and the first I came across were Freenet6, who are linked to Canarie, the Canadian Academic & Research Network. Later I discovered that Belnet in Belgium also provide free tunnels to the 6Bone so I also connected with them.

point link, i.e. the other end of the tunnel. There are also a number of other parameters that can be set in addition to the ones contained in the main syntax. They come after the main parameters have been set. A list of some of the parameters is as follows:

- alias: establishes an additional network address for the interface
- anycast: (for inet6 only) to replace the broadcast address parameter for IPv4
- arp: address resolution protocol
- broadcast: specifies the broadcast address for the interface
- down: used to mark an interface as 'down', i.e. messages won't be sent there
- mtu: allows you to specify the maximum transmission unit of the interface
- netmask: allows the netmask address to be specified for the interface.
- prefixlen: (for inet6 only) specifies that 'len' bits are reserved for subdividing a network into sub-networks. The 'len' must be an integer, and for syntactical reasons it must be a number between 0 and 128. The value 128 is used in this case as there is only one sub-network (i.e. address)
- up: used to mark an interface as 'up', i.e. allow messages to be sent there.

The parameters *prefixlen*, *alias*, and *up* are all used by the commands above, to establish the IPv6 interfaces.

Finally the **route** command is used, to manually manipulate the routing table. It has the syntax **route [-nqv] command [[modifiers] args]**. The **route** command is not normally needed as a system routing-table management daemon, such as **routed** is normally used to handle the routing-table. There are a number of options available when using this command, such as *–n*, which bypasses attempts to print host and network names symbolically when reporting actions (this can be computationally time consuming), *-v*, which is verbose (i.e. print additional information), and *–q*, which suppresses all outputs.

The route utility also provides the following commands:

- add : Add a route,
- flush: Remove all routes,
- delete: Delete a specific route,
- change: Changes aspects of a route – such as its gateway,
- get: Lookup and display the route for a destination
- monitor: Continually report any changes to the routing information base.

The *add* command is used above to add the inet6 route to the IPv6 address of the host. The perl code written to automate the process of setting up the interfaces and adding

the route to the routing table of the host, uses a variable ‘if’, which is used to detect the next available interface for the information to be written to.

Most providers of the tunnel also have a set up script that does all of this for you. Appendix E contains the perl script provided by Freenet6 to find an available interface, and to establish the tunnel from there. If the tunnel is set up correctly the command **ifconfig** in FreeBSD will allow you to view your interfaces, and you should see your IPv6 address for your machine, a link-local address for your machine (for the purpose of autoconfiguring your IPv6 address), your original IPv4 address for the machine, and you will also see the tunnel connection. Figure 2.12 is the response of the FreeBSD machine to an **ifconfig** request.

The IPv4 interfaces are recognised by the **inet** prefix, and the IPv6 interfaces are listed as the **inet6** interfaces. As can be seen from the output, the machine has the IPv4 address 136.206.25.248, and the IPv6 address of 3ffe:b00:c18:1fff::21f. The link local address is fe80::220:afff:fe20:6bf0, which is recognisable from its first four bytes of the address (fe80), and you can see that it has been autoconfigured using the Machine Address Code (MAC address), ether 00:20:af:20:6b:f0. The other end of the tunnel is 3ffe:b00:c18:1fff::21e, and the tunnel connection can be seen on the generic tunnel interface (gif0). As with IPv4 only configurations, there are loopback addresses (localhosts) for both IPv4 and IPv6; lo0 is the interface they can be seen on.

```
hobbes#ifconfig
ep0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        inet 136.206.25.248 netmask 0xffffff00 broadcast 136.206.25.255
        inet6 fe80::220:afff:fe20:6bf0%ep0 prefixlen 64 scopeid 0x1
        ether 00:20:af:20:6b:f0
        media: 10base2/BNC
        supported media: 10base2/BNC 10baseT/UTP 10base5/AUI
gif0: flags=8011<UP,POINTOPOINT,MULTICAST> mtu 1280
        inet6 fe80::220:afff:fe20:6bf0%gif0 --> :: prefixlen 64 scopeid 0x2
        inet6 3ffe:b00:c18:1fff::21f --> 3ffe:b00:c18:1fff::21e prefixlen 128
gif1: flags=8010<POINTOPOINT,MULTICAST> mtu 1280
gif2: flags=8010<POINTOPOINT,MULTICAST> mtu 1280
gif3: flags=8010<POINTOPOINT,MULTICAST> mtu 1280
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 16384
        inet6 fe80::1%lo0 prefixlen 64 scopeid 0x6
        inet6 ::1 prefixlen 128
        inet 127.0.0.1 netmask 0xff000000
ppp0: flags=8010<POINTOPOINT,MULTICAST> mtu 1500
sl0: flags=c010<POINTOPOINT,LINK2,MULTICAST> mtu 552
faith0: flags=8000<MULTICAST> mtu 1500
hobbes#exit
```

Figure 3.2 Output of **ifconfig** command

The show network status command, **netstat**, can be used to show all the interfaces. Using

netstat – rn shows the routing table, and lists network addresses as numbers, giving a clear output of all the established IPv4 and IPv6 addresses on the machine. Figure 2.14 shows the output of this.

```
hobbes#netstat –rn
Routing tables

Internet:
Destination        Gateway            Flags     Refs     Use     Netif Expire
default            136.206.25.254     UGSc         3     1708      ep0
127.0.0.1          127.0.0.1          UH           0      104      lo0
136.206.25/24      link#1             UC           0        0      ep0 =>
136.206.25.248     0:20:af:20:6b:f0   UHLW         0      414      lo0
136.206.25.249     0:20:af:ac:6e:8f   UHLW         1     2194      ep0   1153
136.206.25.254     0:80:21:a:c0:15    UHLW         3        0      ep0   1128

Internet6:
Destination                       Gateway                       Flags      Netif
Expire
::/96                             ::1                           UGRSc       lo0 =>
default                           3ffe:b00:c18:1fff::21f        UGSc       gif0
::1                               ::1                           UH          lo0
::ffff:0.0.0.0/96                 ::1                           UGRSc       lo0
3ffe:b00:c18:1fff::21e            3ffe:b00:c18:1fff::21f        UH         gif0
3ffe:b00:c18:1fff::21f            ::1                           UH          lo0
fe80::/10                         link#1                        UCS         ep0
fe80::%ep0/64                     link#1                        UC          ep0
fe80::%gif0/64                    link#2                        UC         gif0
fe80::220:afff:fe20:6bf0%gif0     ::1                           UH          lo0
fe80::%lo0/64                     fe80::1%lo0                   Uc          lo0
ff01::/32                         ::1                           U           lo0
ff02::/16                         link#1                        UCS         ep0
ff02::%ep0/32                     link#1                        UC          ep0
ff02::%gif0/32                    link#2                        UC         gif0
ff02::%lo0/32                     fe80::1%lo0                   UC          lo0
```

Figure 3.3 Output of **netstat –rn** command, showing the routing table of the FreeBSD machine

The **faith** interface is a special interface that captures IPv6 TCP traffic for implementing user end IPv6-to-IPv4 TCP relays like **faithd**[13]. [11] When IPv6 TCP traffic is seen on a router

(or host functioning as a router), the routing table suggests routing it to the **faith** interface. The packet will be accepted by the router, regardless of the list of IPv6 interface addresses

[13] The FAITH IPv6/v4 translator daemon provides an IPv6-to-IPv4 TCP relay service. It must be used on an IPv4/v6 dual-stack router.
http://www.freebsd.org/cgi/man.cgi?query=faithd&sektion=8&apropos=0&manpath=FreeBSD+4.2-RELEASE

assigned to the router, and will be captured by an IPv6 TCP socket if it has the IN6P_FAITH flag turned on and it has matching address/port pairs.

As a result, **faith** will let you divert IPv6 TCP traffic to some specific destination addresses. **Faithd** can use this behaviour to relay IPv6 TCP traffic to IPv4 TCP traffic. The program can accept some specific IPv6 TCP traffic, perform **getsockname**[14] to get the IPv6 destination address specified by the client, and perform application-specific address mapping to relay IPv6 TCP to IPv4 TCP. Faith is used to implement the 'encapsulation process' described in section 2.3.2, in order to tunnel IPv6 traffic through the IPv4 network.

[14] C command that's used to 'get socket name'

3.3.2 On Windows2000

Again, for the purpose of demonstrating how the tunnel was set up, it's assumed that the IPv6 stack has been installed already. Freenet6 also provided me with a tunnel to the 6Bone for the Windows 2000 machine. The set up for this was similar to that of the FreeBSD machine, but in this case they provided a batch file to create the interfaces and to establish the tunnel (see Appendix F for this file).

There are two DOS commands for doing this [10]:

- `ipv6 rtu ::/0 2/::206.123.31.102 pub`
- `ipv6 adu 2/3ffe:b00:c18:1fff:0:0:0:193`

The first one, **ipv6 rtu**, is used to add or remove a route from the routing table. This command has a number of different arguments to be set (most of which are optional), like the route prefix, lifetime and preference.

The route prefix is not optional. The prefix can either be on-link to a specified interface, or it can be next-hop - specified with a neighbour address on an interface. In this case it's the IPv4 address of the server the Freenet6 side of the tunnel. The prefix is embedded in an IPv6 address (by appended leading zeros to make it compatible). The route can have a lifetime expressed in units of seconds (the default is infinite), and a preference (the default is zero, or most preferred). Specifying a lifetime of zero causes the route to be deleted.

If the route is specified as published (i.e. that the next-hop address is followed by **pub**) meaning it will be used in constructing router advertisements, then by default it does not age. The route's lifetime does not decrement, so it is effectively infinite, but the value set for that argument is used in router advertisements. Otherwise the route can be specified as a published route that ages; a non-published route always ages by default. When using this command, you can abbreviate lifetime as **life**, preference as **pref**, and publish as **pub**. So the configuration sets the next-hop IP as ::206.123.31.102 on interface 2, and the route is used in constructing router advertisements (to 'announce' the tunnel at the Freenet6 interface).

The second command, **ipv6 adu**, is used to add or remove a unicast or anycast address assignment on an interface, defaulting to unicast, if neither is specified. It also takes the option of lifetime, which can be abbreviated to **life** again; the default value is also infinity. So the configuration sets the IPv6 address (3ffe:b00:c18:1fff::193) on interface 2.

The batch file supplied by Freenet6 also logs the date and the time to a file, to show when the tunnel batch file was run last.

In order to check to see if your set up has been completed, the DOS command **ipv6 if** shows the interfaces that have been established. Figure 2.13 shows an output of the configuration[15].

```
C:\>ipv6 if
Interface 4 (site 1): Local Area Connection
  uses Neighbor Discovery
  link-level address: 00-20-af-ac-6e-8f
    preferred address fe80::220:afff:feac:6e8f, infinite/infinite
    multicast address ff02::1, 1 refs, not reportable
    multicast address ff02::1:ffac:6e8f, 1 refs, last reporter
  link MTU 1500 (true link MTU 1500)
  current hop limit 128
  reachable time 32000ms (base 30000ms)
  retransmission interval 1000ms
  DAD transmits 1
Interface 3 (site 1): 6-over-4 Virtual Interface
  uses Neighbor Discovery
  link-level address: 136.206.25.249
    preferred address fe80::88ce:19f9, infinite/infinite
    multicast address ff02::1, 1 refs, not reportable
    multicast address ff02::1:ffce:19f9, 1 refs, last reporter
  link MTU 1280 (true link MTU 65515)
  current hop limit 128
  reachable time 21500ms (base 30000ms)
  retransmission interval 1000ms
  DAD transmits 1
Interface 2 (site 0): Tunnel Pseudo-Interface
  does not use Neighbor Discovery
  link-level address: 0.0.0.0
    preferred address 3ffe:b00:c18:1fff::193, infinite/infinite
    preferred address ::136.206.25.249, infinite/infinite
  link MTU 1280 (true link MTU 65515)
  current hop limit 128
  reachable time 0ms (base 0ms)
  retransmission interval 0ms
  DAD transmits 0
Interface 1 (site 0): Loopback Pseudo-Interface
```

[15] The tunnels should be not be created behind a firewall**.** When I first set up the tunnels I was behind the Universities firewall, which caused undetermined problems for the Windows 2000 machines' tunnel, but the FreeBSD one was fine. So I had to establish a second tunnel for the Windows 2000 machine.

```
  does not use Neighbor Discovery
  link-level address:
    preferred address ::1, infinite/infinite
  link MTU 1500 (true link MTU 1500)
  current hop limit 1
  reachable time 0ms (base 0ms)
  retransmission interval 0ms
  DAD transmits 0

C:\>
```

Figure 3.4 Output of **ipv6 if** command.

If an interface number is specified, it will only show the information for that interface. By checking interface number two, the tunnel pseudo interface, we see that the commands have been successful and the ends of the tunnels have been created. The link-layer address is also listed on interface 4, and it consists of the MAC address as with the FreeBSD machine - fe80::220:afff:feac:6e8f. If the link layer address was of the format a.b.c.d then it would be a 6-over-4 interface.

Interface one is a pseudo interface used for loopback (i.e. the local host). Interface two is a pseudo-interface used for configured tunnels, automatic tunnels and 6-over-4 tunnelling. Other interfaces are numbered in increasing order from here, as they are added.

Chapter 4

Testing and Benchmarking

To test IPv6 it was decided to approach it from the point of view of the user, and not the administrator[16]. That is, what applications are supported by IPv6 that are currently widely used by the Internet community at large, and how well are they supported for the two platforms – FreeBSD and Windows 2000. The testing was broken down into two main sections, Internet Control Message Protocol (ICMP) testing and Transmission Control Protocol (TCP) testing (i.e. based on which IPv6 dependent protocol they use).

4.1 Internet Control Message Protocol (ICMP)

ICMP is the control message protocol used by the Internet. It is probably most recognised in its form of **ping** and **traceroute**, which are generally used to check the status of remote hosts, and the 'distance' they are away.

4.1.1 Ping 6

Ping uses timed IP/ICMP ECHO_REQUEST and ECHO_REPLY packets to probe the "distance" to a target machine. It is generally used today to test quickly whether a machine is "alive" or not, remotely. Unfortunately, due to the misuse of ping to cause Denial of Service attacks on networks, blocking ICMP traffic at the router is now a frequent occurrence, so it's not as reliable as it used to be.

[16] I chose this based on the resources for this project, as much as I would have liked to set up IPv6 DNS and routers, I was limited in what I could do, and because the usability of IPv6 will be the true test of whether the change will be swift and painless or not.

Ping6 is the IPv6 version of **ping**, and it comes as a standard package with the FreeBSD/KAME IPv6 stack, and the Microsoft Research IPv6 stack. To test if the tunnels were active after setting them up – a **ping6** command was sent to the other end of the tunnel. The results for the FreeBSD machine and windows machine are in Figure 4.1 and 4.2:

```
C:\>ping6 3ffe:b00:c18:1fff::912

Pinging 3ffe:b00:c18:1fff::912 with 32 bytes of data:

Reply from 3ffe:b00:c18:1fff::912: bytes=32 time=164ms

Reply from 3ffe:b00:c18:1fff::912: bytes=32 time=146ms

Reply from 3ffe:b00:c18:1fff::912: bytes=32 time=142ms

Reply from 3ffe:b00:c18:1fff::912: bytes=32 time=150ms

C:\>
```

Figure 4.1 Ping response from the other end of the Windows 2000 tunnel

```
hobbes#ping6 3ffe:b00:c18:1fff::21e

PING6(56=40+8+8 bytes) 3ffe:b00:c18:1fff::21f -->3ffe:b00:c18:1fff::21e

16 bytes from 3ffe:b00:c18:1fff::21e, icmp_seq=0 hlim=64 time=159.256 ms

16 bytes from 3ffe:b00:c18:1fff::21e, icmp_seq=1 hlim=64 time=141.719 ms

16 bytes from 3ffe:b00:c18:1fff::21e, icmp_seq=2 hlim=64 time=148.301 ms

16 bytes from 3ffe:b00:c18:1fff::21e, icmp_seq=3 hlim=64 time=142.68 ms

16 bytes from 3ffe:b00:c18:1fff::21e, icmp_seq=4 hlim=64 time=144.599 ms

16 bytes from 3ffe:b00:c18:1fff::21e, icmp_seq=5 hlim=64 time=140.075 ms

16 bytes from 3ffe:b00:c18:1fff::21e, icmp_seq=6 hlim=64 time=141.776 ms

16 bytes from 3ffe:b00:c18:1fff::21e, icmp_seq=7 hlim=64 time=144.261 ms

16 bytes from 3ffe:b00:c18:1fff::21e, icmp_seq=8 hlim=64 time=147.628 ms

^C

--- 3ffe:b00:c18:1fff::21e ping6 statistics ---

9 packets transmitted, 9 packets received, 0% packet loss

round-trip min/avg/max = 140.057/145.588/159.256 ms
hobbes#exit
```

Figure 4.2 Ping results from the other end of the FreeBSD tunnel

The ping program is implemented differently on different platforms, and as ping was developed by Berkeley Unix (FreeBSD) it's version offers a lot more than the Windows 2000 version does, such as ping statistics, hop limit and numbering of hops using **icmp_seq** (useful for seeing which packet was dropped, if any). In FreeBSD the ping will continue for the maximum number of hop limits allowed (this varies from site to site), but the Windows 2000 version only sends four ICMP requests. That's why the ping on the FreeBSD machine was cancelled, and it claims that it has lost packets because of this. Figure 4.3 shows some ping outputs to overseas servers that have IPv6 addresses, and that allow ICMP traffic, on the Windows 2000 machine.

```
C:\>ping6 www.normos.org

Pinging www.normos.org [3ffe:b00:c18:1::11] with 32 bytes of data:

Reply from 3ffe:b00:c18:1::11: bytes=32 time=141ms
Reply from 3ffe:b00:c18:1::11: bytes=32 time=142ms
Reply from 3ffe:b00:c18:1::11: bytes=32 time=139ms
Reply from 3ffe:b00:c18:1::11: bytes=32 time=139ms

C:\>

C:\>ping6 ipv6.research.microsoft.com

Pinging  ipv6.research.microsoft.com  [2002:836b:4179::836b:4179]  with  32
bytes of data:

Reply from 2002:836b:4179::836b:4179: bytes=32 time=500ms
Reply from 2002:836b:4179::836b:4179: bytes=32 time=509ms
Reply from 2002:836b:4179::836b:4179: bytes=32 time=510ms
Reply from 2002:836b:4179::836b:4179: bytes=32 time=522ms

C:\>

C:\>ping6 www.kame.net

Pinging kame212.kame.net [2001:200:0:4819:5054:ff:fedc:50d2] with 32 bytes
of
data:

Reply from 2001:200:0:4819:5054:ff:fedc:50d2: bytes=32 time=673ms
Reply from 2001:200:0:4819:5054:ff:fedc:50d2: bytes=32 time=664ms
Reply from 2001:200:0:4819:5054:ff:fedc:50d2: bytes=32 time=656ms
Reply from 2001:200:0:4819:5054:ff:fedc:50d2: bytes=32 time=670ms

C:\>
```

Figure 4.3 Ping responses from IPv6 servers taken on the Windows 2000 machine

Figure 4.4 shows ping responses to overseas servers that have IPv6 addresses, and that allow ICMP traffic from the FreeBSD machine.

```
hobbes#ping6 www.6tap.net
PING6(56=40+8+8 bytes) 3ffe:b00:c18:1fff::21f --> 3ffe:700:20:4:200:f8ff:fe75:6bc7
16 bytes from 3ffe:700:20:4:200:f8ff:fe75:6bc7, icmp_seq=0 hlim=57 time=305.174 ms
16 bytes from 3ffe:700:20:4:200:f8ff:fe75:6bc7, icmp_seq=1 hlim=57 time=293.443 ms
16 bytes from 3ffe:700:20:4:200:f8ff:fe75:6bc7, icmp_seq=2 hlim=57 time=288.396 ms
```

```
16 bytes from 3ffe:700:20:4:200:f8ff:fe75:6bc7, icmp_seq=3 hlim=57 time=327.651 ms
16 bytes from 3ffe:700:20:4:200:f8ff:fe75:6bc7, icmp_seq=4 hlim=57 time=291.942 ms
16 bytes from 3ffe:700:20:4:200:f8ff:fe75:6bc7, icmp_seq=5 hlim=57 time=290.243 ms
16 bytes from 3ffe:700:20:4:200:f8ff:fe75:6bc7, icmp_seq=6 hlim=57 time=287.575 ms
16 bytes from 3ffe:700:20:4:200:f8ff:fe75:6bc7, icmp_seq=7 hlim=57 time=306.549 ms
16 bytes from 3ffe:700:20:4:200:f8ff:fe75:6bc7, icmp_seq=8 hlim=57 time=299.669 ms
16 bytes from 3ffe:700:20:4:200:f8ff:fe75:6bc7, icmp_seq=9 hlim=57 time=292.464 ms
^C
--- www.6tap.net ping6 statistics ---
11 packets transmitted, 10 packets received, 9% packet loss
round-trip min/avg/max = 287.575/298.31/327.651 ms

hobbes#ping6 www.zamanetworks.com
PING6(56=40+8+8 bytes) 3ffe:b00:c18:1fff::21f --> 3ffe:80f0:1:1:201:2ff:fee8:efa1
16 bytes from 3ffe:80f0:1:1:201:2ff:fee8:efa1, icmp_seq=0 hlim=250 time=669.285 ms
16 bytes from 3ffe:80f0:1:1:201:2ff:fee8:efa1, icmp_seq=1 hlim=250 time=671.243 ms
16 bytes from 3ffe:80f0:1:1:201:2ff:fee8:efa1, icmp_seq=2 hlim=250 time=682.505 ms
16 bytes from 3ffe:80f0:1:1:201:2ff:fee8:efa1, icmp_seq=3 hlim=250 time=683.961 ms
16 bytes from 3ffe:80f0:1:1:201:2ff:fee8:efa1, icmp_seq=4 hlim=250 time=658.956 ms
16 bytes from 3ffe:80f0:1:1:201:2ff:fee8:efa1, icmp_seq=5 hlim=250 time=679.692 ms
16 bytes from 3ffe:80f0:1:1:201:2ff:fee8:efa1, icmp_seq=6 hlim=250 time=657.484 ms
16 bytes from 3ffe:80f0:1:1:201:2ff:fee8:efa1, icmp_seq=7 hlim=250 time=670.935 ms
16 bytes from 3ffe:80f0:1:1:201:2ff:fee8:efa1, icmp_seq=8 hlim=250 time=687.593 ms
16 bytes from 3ffe:80f0:1:1:201:2ff:fee8:efa1, icmp_seq=9 hlim=250 time=676.625 ms
^C--- www.zamanetworks.com ping6 statistics ---
11 packets transmitted, 10 packets received, 9% packet loss
round-trip min/avg/max = 657.484/673.827/687.593 ms

hobbes#ping6 www.kame.net
PING6(56=40+8+8            bytes)            3ffe:b00:c18:1fff::21f            -->
2001:200:0:4819:5054:ff:fedc:50d2
16 bytes from 2001:200:0:4819:5054:ff:fedc:50d2, icmp_seq=0 hlim=53 time=686.833
ms
16 bytes from 2001:200:0:4819:5054:ff:fedc:50d2, icmp_seq=1 hlim=53 time=657.283
ms
16 bytes from 2001:200:0:4819:5054:ff:fedc:50d2, icmp_seq=2 hlim=53 time=659.879
ms
16 bytes from 2001:200:0:4819:5054:ff:fedc:50d2, icmp_seq=3 hlim=53 time=665.104
ms
16 bytes from 2001:200:0:4819:5054:ff:fedc:50d2, icmp_seq=4 hlim=53 time=659.384
ms
16 bytes from 2001:200:0:4819:5054:ff:fedc:50d2, icmp_seq=5 hlim=53 time=658.44 ms
16 bytes from 2001:200:0:4819:5054:ff:fedc:50d2, icmp_seq=6 hlim=53 time=665.532
ms
16 bytes from 2001:200:0:4819:5054:ff:fedc:50d2, icmp_seq=7 hlim=53 time=658.84 ms
16 bytes from 2001:200:0:4819:5054:ff:fedc:50d2, icmp_seq=8 hlim=53 time=657.409
ms
^C
--- www.kame.net ping6 statistics ---
10 packets transmitted, 9 packets received, 10% packet loss
round-trip min/avg/max = 657.283/663.189/686.833 ms
hobbes#exit
```

Figure 4.4 Ping responses from IPv6 servers taken on the FreeBSD machine

4.1.2 Traceroute 6

Traceroute uses ICMP Time-to-Live Exceeded messages when pinging by modulating the IP time to life (TTL) header field, to mark out the number of hops between the source address and the destination address of the packet. This has also been modified for IPv6 in the form of **traceroute6** (**tracert6** on Windows 2000). Unlike ping, traceroute does not generally get blocked at the router. Figure 4.5 shows the traceroutes from the Windows machine to a number of IPv6 sites. The asterisks that appear in the traceroute occur when a response to the packet is not received.

```
C:\>tracert6 www.kame.net

Tracing route to kame212.kame.net [3ffe:501:4819:2000:5054:ff:fedc:50d2]
over a maximum of 30 hops:

1   281 ms   252 ms   244 ms  3ffe:b00:c18:1fff::192
2   240 ms   265 ms   248 ms  rap.ipv6.viagenie.qc.ca
   [3ffe:b00:c18:1:290:27ff:fe17:fc0f]
3   586 ms     *      693 ms  3ffe:8000:ffff:b::1
4   886 ms   771 ms     *     3ffe:8000:ffff:5::2
5   832 ms   782 ms   785 ms  pc7.otemachi.wide.ad.jp
   [3ffe:501:0:1802:2e0:18ff:fe98:a28d]
6   783 ms   789 ms   769 ms  pc2.fujisawa.wide.ad.jp
   [2001:200:0:1001:2a0:24ff:fe83:8b33]
7   777 ms   775 ms   744 ms  paradise.v6.kame.net
   [3ffe:501:4819:2000:2e0:18ff:fe98:f19d]
8   755 ms   763 ms   771 ms  pine.v6.kame.net
   [3ffe:501:4819:2000:5054:ff:fedc:50d2]

Trace complete.

C:\>tracert6 www.normos.org

Tracing route to www.normos.org [3ffe:b00:c18:1::11]
over a maximum of 30 hops:

  1     *      264 ms     *     www.normos.org [3ffe:b00:c18:1::11]
  2     *      279 ms   256 ms  www.normos.org [3ffe:b00:c18:1::11]

Trace complete.

C:\>tracert6 www.6tap.net

Tracing route to www.6tap.net [3ffe:700:20:4:200:f8ff:fe75:6bc7]
over a maximum of 30 hops:

  1   295 ms   314 ms   294 ms  3ffe:b00:c18:1fff::192
  2   317 ms   309 ms   286 ms  rap.ipv6.viagenie.qc.ca
     [3ffe:b00:c18:1:290:27ff:fe17:fc0f]
  3   307 ms   329 ms     *     3ffe:b00:c18::f
  4   431 ms   406 ms   426 ms  3ffe:2900:a:4::2
  5   421 ms   427 ms   435 ms  www-6tap.es.net [3ffe:700:20:4:200:f8ff:fe75:6bc7]

Trace complete.

C:\>tracert6 www.zamanetworks.com

Tracing route to www.zamanetworks.com [3ffe:80f0:1:1:201:2ff:fee8:efa1]
over a maximum of 30 hops:

  1   139 ms   134 ms   136 ms  3ffe:b00:c18:1fff::192
  2   145 ms   130 ms   137 ms  rap.ipv6.viagenie.qc.ca
     [3ffe:b00:c18:1:290:27ff:fe17:fc0f]
  3   274 ms   275 ms   270 ms  viagenie-gw.ipv6.wilbury.sk [3ffe:80e1:8000::c]
```

```
  4   480 ms   485 ms   473 ms  3ffe:80e1:8000::13
  5   475 ms   481 ms   485 ms  3ffe:80f0:1:1:201:2ff:fee8:efa1

Trace complete.

C:\>tracert6 ipv6.research.microsoft.com

Tracing route to ipv6.research.microsoft.com [2002:836b:4179::836b:4179]
over a maximum of 30 hops:

  1   144 ms   150 ms   142 ms  3ffe:b00:c18:1fff::192
  2   145 ms   141 ms   148 ms  rap.ipv6.viagenie.qc.ca
     [3ffe:b00:c18:1:290:27ff:fe17:fc0f]
  3   334 ms   354 ms   337 ms  grnet-Viaginie.ipv6.grnet.gr [3ffe:2d00:1::9]
  4   413 ms   417 ms   437 ms  3ffe:80c0:200:5::1d
  5   510 ms   513 ms   492 ms  2002:836b:4179::836b:4179

Trace complete.

C:\>
```

Figure 4.5 Traceroutes to a number of IPv6 machines worldwide.

The traceroute first reaches the other end of the tunnel, then it goes to Viagéne to which Freenet6 is connected to (and who presumably supply their Internet connection), and from there it goes to the Internet, and on to its final destination. The address allocation policy can be seen in practice from this example, as servers in America are reached fairly quickly after the Canadian ones, but the Japanese servers take 8 hops to reach their destination.

Ideally a private IPv6 network should be established, with one machine acting as a gateway to the IPv4 network, and over a tunnel to the IPv6 6Bone in order to compare the traceroute results. This wasn't possible in the case due to time and resources constraints. Also it would be difficult to compare the results unless it was over a very long period of time, as server reliability and uptime, and load averages on Internet connections all factor into the comparison. From the point of view of the user, these applications work just as well (if not exactly the same) as the IPv4 versions.

4.2 Transmission Control Protocol (TCP) testing

This section describes the testing I performed on the machines; to show how IPv6 affects some of the more common TCP based applications that are in frequent use on the Internet

4.2.1 Hyper Text Transfer Protocol (http)

The test for http compatibility with IPv6 was based on the web server and Internet browser compatibility. For this the Apache web server was established on both the FreeBSD machine and the Windows 2000 machine.

Apache has been 'patched'[17] so that it now listens for http connections on port 80 from IPv6 as well as IPv4 addresses. The configuration file contains all the information on setting up the Apache web server, and the content is standard across all platforms. In it, the listen command is set as follows:

```
# Listen: Allows you to bind Apache to specific IP addresses and/or
# ports, in addition to the default. See also the <VirtualHost>
# directive.
# Listen can take two arguments.
# (this is an extension for supporting IPv6 addresses)
Listen :: 80
Listen 0.0.0.0 80
```

The command '`Listen :: 80`', listens for traffic from all IPv6 addresses and is recognised by the hexadecimal colon format. The port, which it "listens" on, is port 80, which is the standard http port. The command '`Listen 0.0.0.0 80`' tells the web server to listen for traffic from all

IPv4 addresses on port 80, and is recognised by its dotted-decimal structure. Using these commands, the web server can be instructed to listen for either IPv6 or IPv4 traffic, or both.

Obviously only the most up-to-date versions of browsers will be IPv6 compatible. Currently Internet Explorer (IE) 5 (and newer versions) is the only graphic browser that's IPv6 compliant for Windows 2000, (although projects are being worked on to make Netscape 6 compliant, [12] but it is poorly documented and I was unable to get it working). This means that they should be able to take the IPv6 address in the location bar, and locate the page as if it was an IPv4 address. The browsers use the 'Wininet.dll' dynamic link library on Windows to allow web browsers to access IPv6 enabled web browsers.

---

[17] http://www.apache.org

Internet explorer (the browser I use on the Windows 2000 machine) can then download web pages if:

- The Domain Name System (DNS) query for the name of the Web server in the URL returns an IPv6 address,
- The URL is the literal format of an IPv6 address. If this is the case, then the IPv6 address must be enclosed in square brackets in order for the browser to reach the web server it's looking for. For example: http://[3ffe:b00:c18:1fff:0:0:0:193]/home.html

Figure 4.6 uses the IPv6 address from the Windows 2000 machine to locate the web page served on that machine by the Apache web server running on that machine.

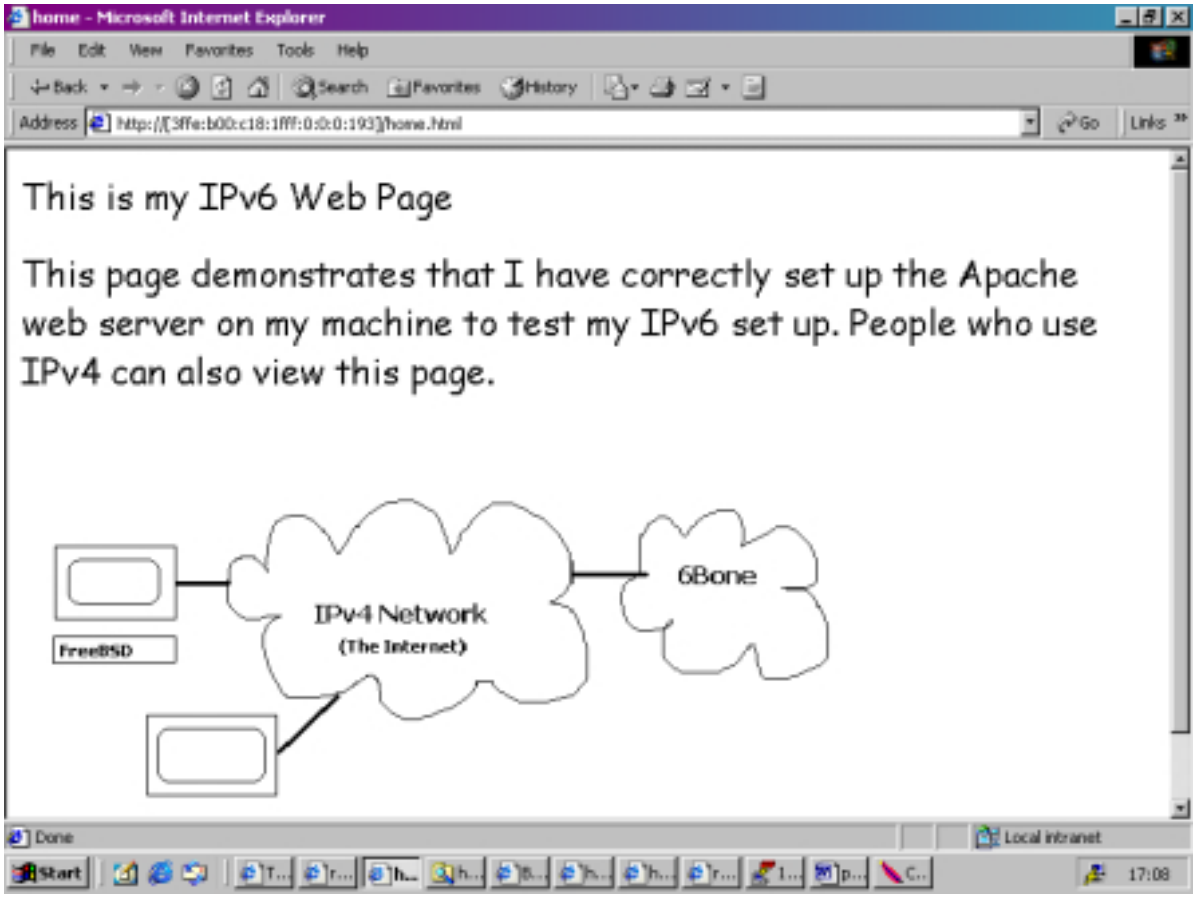


Figure 4.6 Screen shot of browsing the Internet using IPv6 on the Windows 2000 machine.

On FreeBSD, the Mozilla web browser has supposedly been patched for IPv6, by the KAME project. After a week and a half long installation and building of the program from the source code, it has failed to produce any results. It is possible that the browser needs to run on more powerful hardware, but it takes on average ten to fifteen minutes just to load the

browser and even then it does not start up successfully. Unfortunately I have been unable to provide working proof of a graphical browser for IPv6 on FreeBSD. However, there is a text-based browser for FreeBSD (and most Unix operating systems) called ‘Lynx’ that does load IPv6 web pages with ease. To demonstrate the working IPv6 capable Apache Web server hosted on the FreeBSD machine, figure 4.7 contains a screen shot of a web page served up by Apache on the FreeBSD machine, viewed in Internet Explorer.

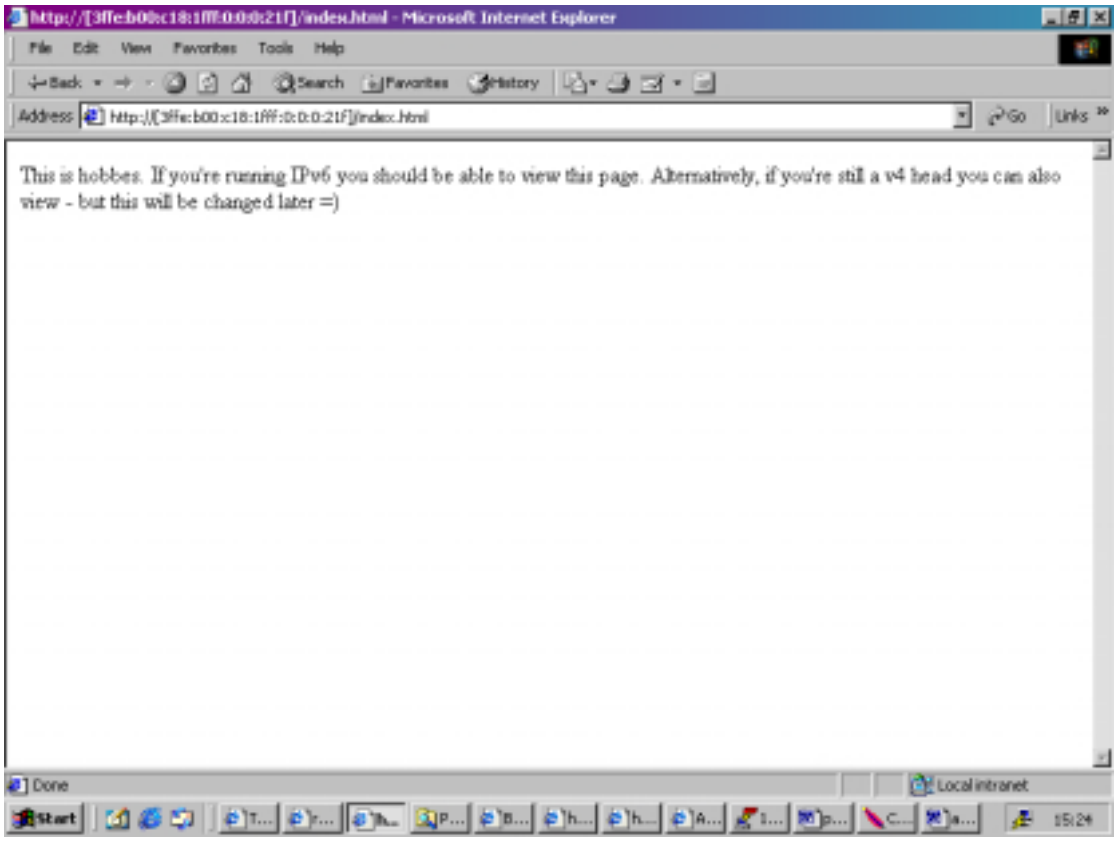


Figure 4.7 Screen shot of a web page served by Apache on the FreeBSD machine.

Of course it would be much easier to view web pages if they had a name associated with their IPv6 address, as with IPv4. The Windows 2000 machine is called “calvin”, and the FreeBSD machine is called “hobbes”. As the machines are on the Electronic Engineering department’s subnet, I could not set up my own DNS for IPv6 to do this (this is described in more detail in Chapter 5), though the machines are entered in the DNS under their IPv4 addresses.

There are a number of IPv6 specific accessible web pages on the Internet like http://ipv6.research.microsoft.com and http://www.ipv6.wanadoo.be. Appendix G shows some screen shots, which depict browsing using IPv6. If you look at the status bar at the

bottom of the browser, you can see where the name is being resolved by the DNS to it’s IPv6 address, in order to locate the site.

4.2.2 Telnet and Secure Shell (SSH)

Telnet and SSH are protocols which allow you to connect to a machine to access it remotely. Telnet was developed by ARPANET, and has since been mostly replaced by regular Telnet users, with the more secure SSH (instead of your password being sent as plain text, it is encrypted with an RSA key (Rivest, Shamir, & Adleman (public key encryption technology)), and then transferred across the Internet, so it is a much more secure method of connection as your password is less likely to be intercepted en-route).

In order for remote connections to be allowed to be established on the FreeBSD machine, the Internet "super-server" daemon **inetd** had to be edited to allow for Telnet and SSH connections over IPv6 and IPv4. When the machine was first set up, all these services were turned off to prevent it being attacked. In order to allow these types of connections to the machine, the inet.conf (configuration file) must be edited to include permission for these connections, and then the inet daemon must be stopped and restarted – because the inet.conf file is only read every time the daemon is started. To stop the inet daemon the process has to be killed using the Hang UP (HUP) option (i.e. **kill –HUP <process ID number>**). Figure 4.7 shows the output from the command **netstat** (show network status) - **netstat –a | grep LISTEN**. From this we can see all the services running from **inetd** that Listen for TCP connections (using **| grep LISTEN** shows only the tcp connections, and not all the term emulators running on the machine). It can be clearly seen, that hobbes is listening for SSH and Telnet connections over tcp4 and tcp46 (IPv4 and IPv6).

```
hobbes#netstat -a |grep LISTEN
tcp4       0      0  *.telnet               *.*                    LISTEN
tcp4       0      0  *.http                 *.*                    LISTEN
tcp46      0      0  *.http                 *.*                    LISTEN
tcp6       0      0  *.telnet               *.*                    LISTEN
tcp4       0      0  *.ssh                  *.*                    LISTEN
tcp46      0      0  *.ssh                  *.*                    LISTEN
tcp4       0      0  *.6000                 *.*                    LISTEN
hobbes#exit
```

Figure 4.8 output of **netstat –a |grep LISTEN**

Windows 2000 does not allow for remote system access, it was not possible to test for SSH and Telnet connections from the FreeBSD machine to the Windows machine. However, SSH over IPv6 can still be done from an IPv4 term on the machine as can be seen in Figure 4.9. The new prompt, shows I'm in a new term session.

```
hobbes# ssh -l orly 3ffe:b00:c18:1fff::21f
orly@3ffe:b00:c18:1fff::21f's password:
Last login: Wed Apr 11 11.34:23 from 136.206.25.248
Copyright (c) 1980, 1983, 1986, 1988, 1990, 1991, 1993, 1994
       The Regents of the University of California.  All rights reserved.

FreeBSD 4.3-BETA (GENERIC) #0  Tue Mar 20 16:25:07 GMT 2001

The Days are just Packed!
[orly@hobbes] [~]=>                                                (p4) [10:48]
```

Figure 4.9 Connecting to hobbes over IPv6 from an IPv4 terminal on hobbes

Tera Term Pro, a standard term emulator for windows, has been patched to allow for IPv6 telnet and ssh connections [13]. The protocol selector on the graphical user interface (GUI) allows you to choose which protocol you wish to connect with. As with Internet Explorer, it is necessary to enter the IPv6 address in the host box within square brackets "[]", as it doesn't recognise the hexadecimal colon format otherwise, and will complain that its an 'Invalid Host'.

Figure 4.10 shows the protocol selector, and the IPv6 address being entered just before the connection to hobbes is made, and figure 4.11 shows Tera Term after the connection has been made (recognised by the IPv6 address of hobbes in the place where it says [disconnected] in figure 4.10).

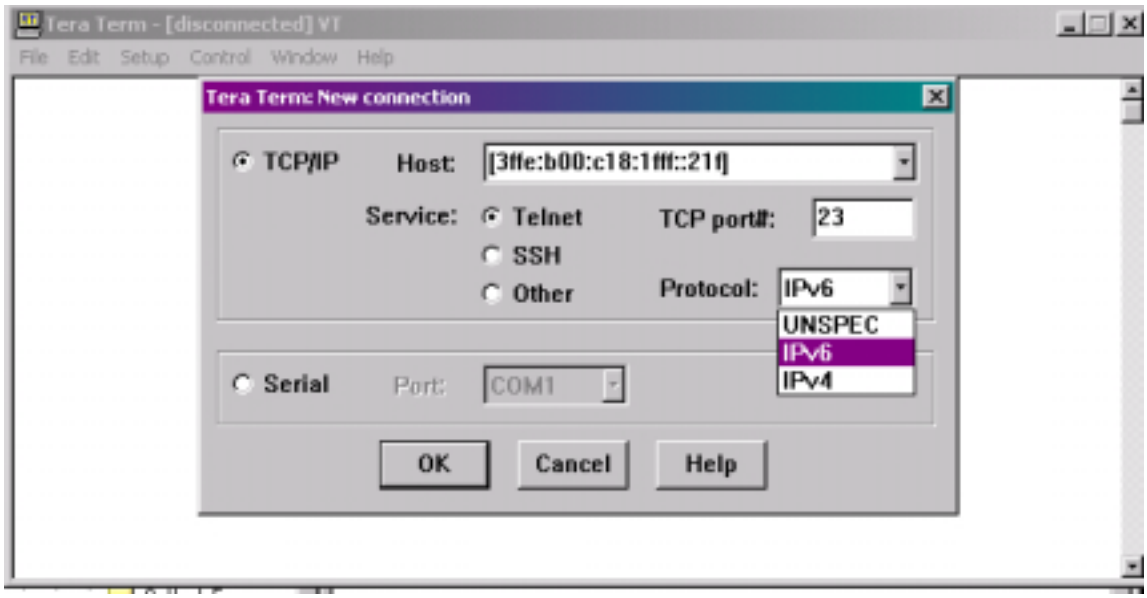


Figure 4.10 Screen shot to show IPv6 and IPv4 choice for Telnet and SSH

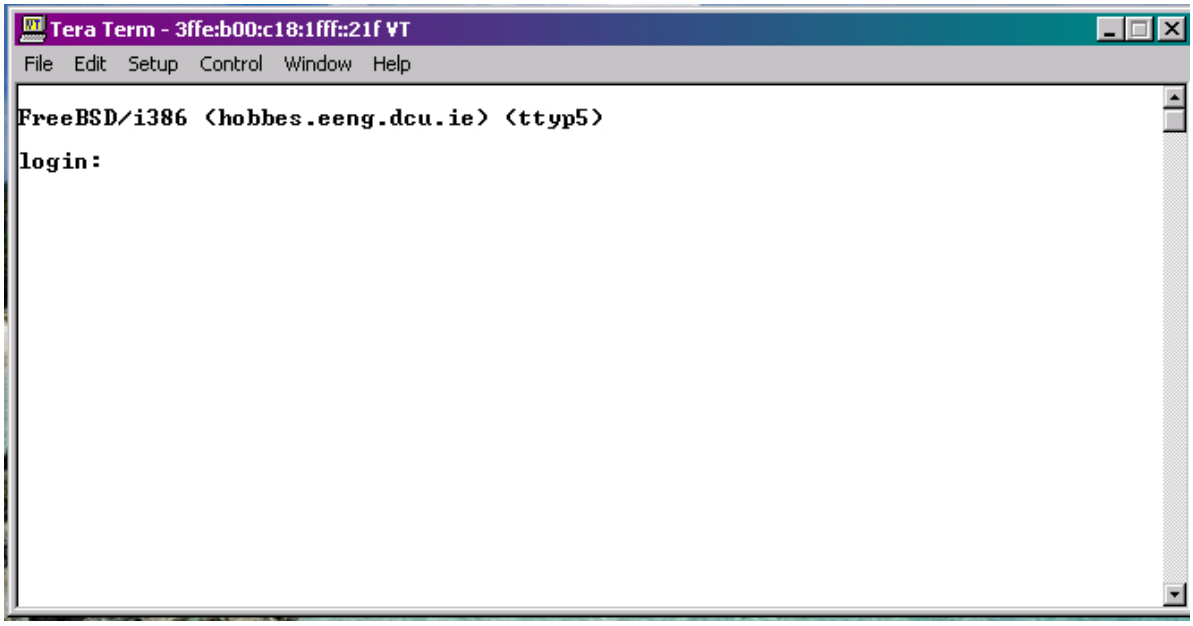


Figure 4.11 Screen shot of an SSH connection to hobbes from calvin.

Another term emulator, which is very popular called PuTTY, has also been patched to work with IPv6 [14]. Unlike Tera Term, this is a completely intuitive term emulator, and you can just enter the IPv6 address in the Host Name field. You don't have to choose whether you're using IPv6 or IPv4 either, the program knows by checking the format of the Host Name (i.e. either dotted-decimal, or hexadecimal colon format). Figure 4.12 shows PuTTYs configuration box, where an IPv6 connection to Hobbes has been established using SSH.

PuTTY also allows you to store session logins, so if you don't have an IPv6 DNS you don't have to keep entering your cumbersome IPv6 address every time you wish to start a term. This is very important for people who make regular remote connections to a host. These are stored in the Saved Sessions Field.

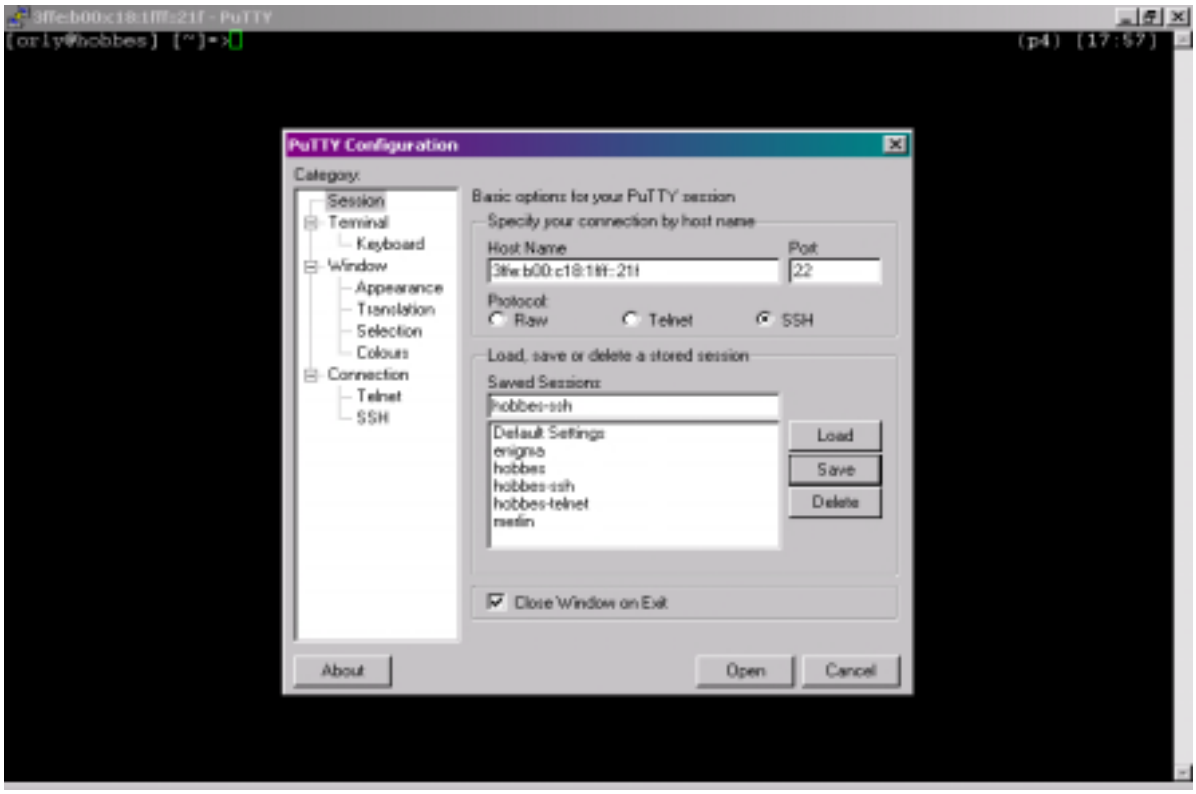


Figure 4.12 Establishing an SSH connection to hobbes over IPv6 using PuTTY

Figure 4.13 shows the SSH connection to hobbes being made. The format for connecting is exactly the same as with IPv4 so it shouldn't pose any problem to users when their system is changed from IPv4 to IPv6.

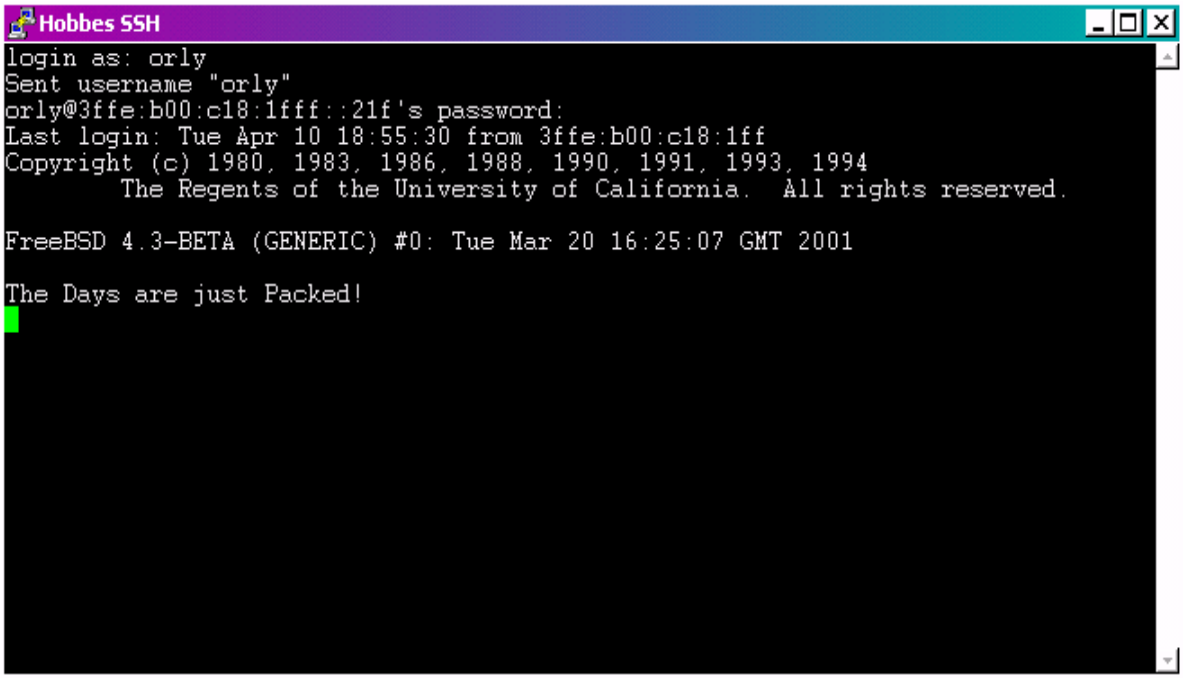


Figure 4.13 Putty Screen shot of SSH connection to hobbes

4.2.3 Internet Relay Chat (IRC)

Internet Relay Chat is probably the most popular application used on the Internet, after the web browser. IRC is the classic peer-to-peer client server system, and is therefore an important test bed application for the transition to IPv6. It is dependent on TCP/IP for the transmission of data across the Internet. Most of the regularly used versions on FreeBSD and other Unix operating systems of IRC are being updated to work under IPv6. The BitchX and Epic IRC clients – two of the more popular Unix based IRC clients, have released a beta version of their software which has been updated to work with IPv6 addresses (this has been done by the KAME project). FreeBSD also has an updated version of the IRC daemon, which allows you to host your own IRC server.

The augmentation of these clients, and the servers, which they connect to are still very much under development, but they are reputed to be working to the point where they can then be used to connect to IPv6 specific IRC servers, like ircnet.wanadoo.be, which is run by the telecom company, Belnet. I attempted to get an IRC server running on the FreeBSD machine, but it could not be determined how to allow connections from IPv6 hosts to the server. The documentation that is provided with these packages for FreeBSD needs to be updated, and made less obfuscated in order to allow for a smoother transition of these services to IPv6.

Attempts to establish a connection to one of the IPv6 IRC servers on the web using BitchX also proved unsuccessful. The servers experienced frequent downtime, and were generally unreliable for testing purposes (at present), but as more are established, it is anticipated that it will become a successful test for the robustness of IPv6. Again documentation about how to connect to the IRC server and the requirements for connecting was not very clear.

The Microsoft IRC client, MIRC has yet to be updated to cope with IPv6 Addresses. There is an IP bouncer[18] available on the World Wide Web that claims to be able to use the IPv4 address of a host to ‘bounce’ data to an IPv6 IRC server, under the guise of an IPv6 address. All attempts to get this working proved unsuccessful. The configuration file was set up from the instructions given, but I still can’t connect to the IPv6 IRC channel, ircnet.wanadoo.be. Figure 4.14 shows a screen shot of the bouncer and it’s configuration

[18] http://tunnel.be.wanadoo.com/bin/AsyBoV6_1_2_0_0.zip

panel, and Figure 4.15 shows a screen shot of the attempts to connect to ircnet.wanadoo.be. Each time the connection is refused cause the IPv6 address doesn't get recognised, only the IPv4 address is.

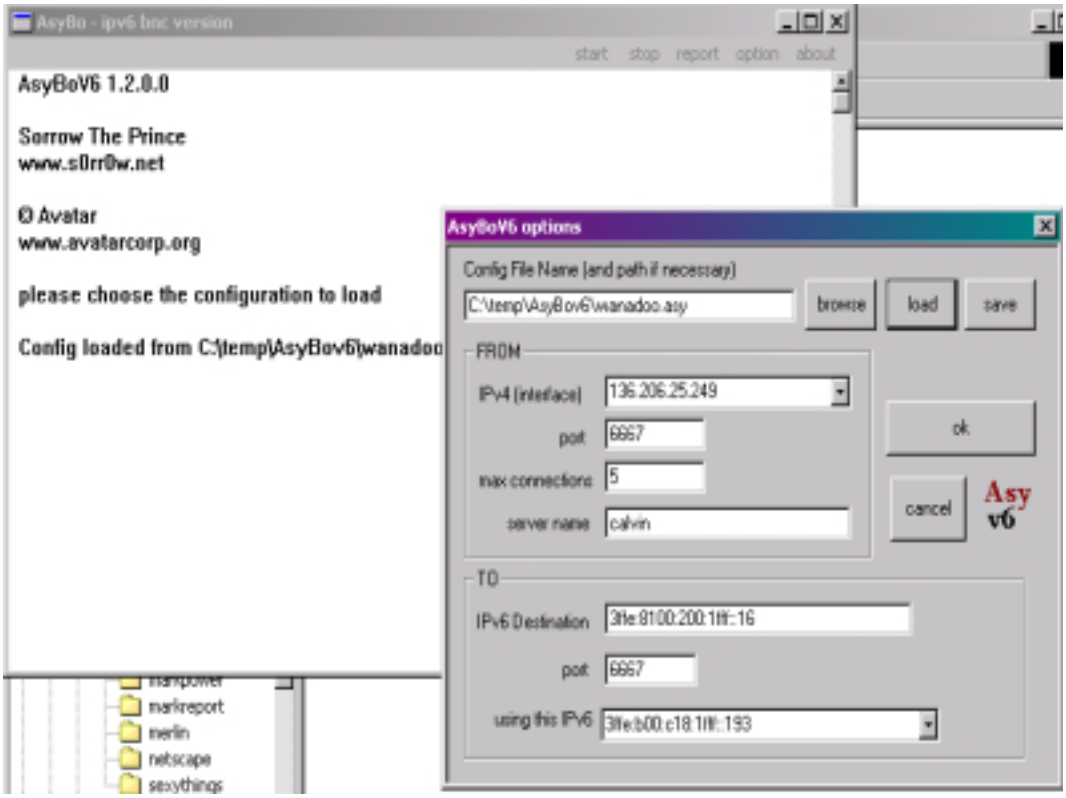


Figure 4.14 Screen shot of the configuration of the IP bouncer

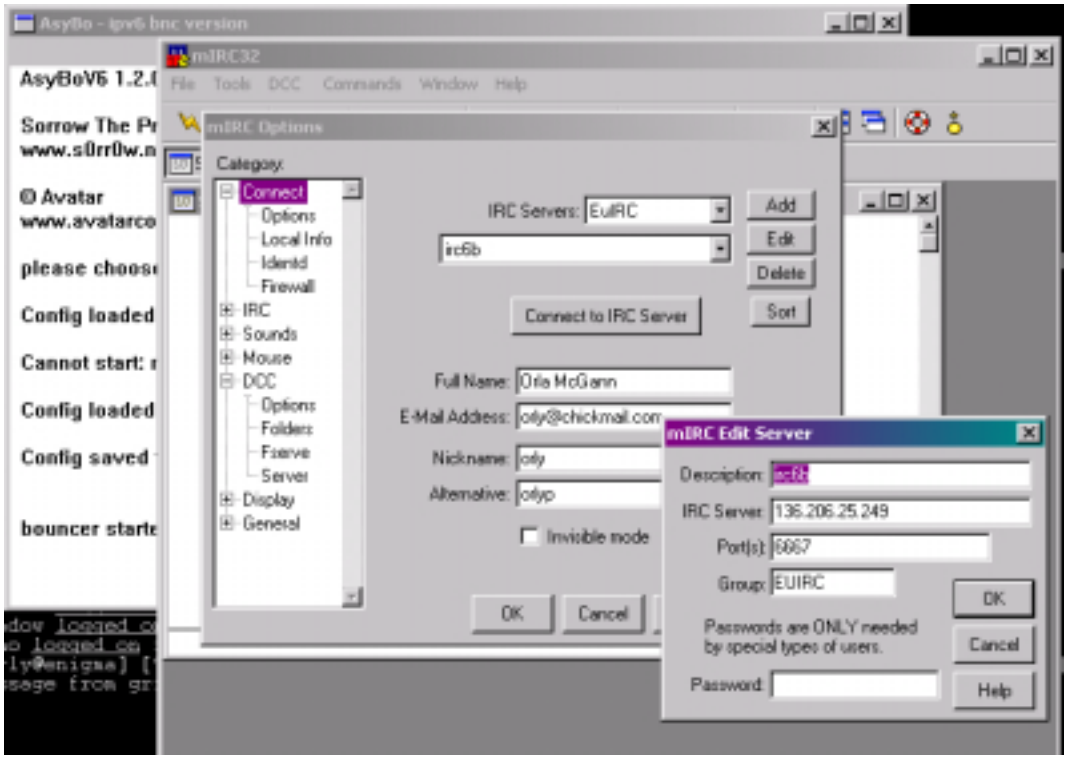


Figure 4.15 Screen shot of the configuration of the MIRC connection

Figure 4.16 shows a screen shot of the Microsoft IRC client trying to connect to ircnet.wanadoo.be using the bouncer.

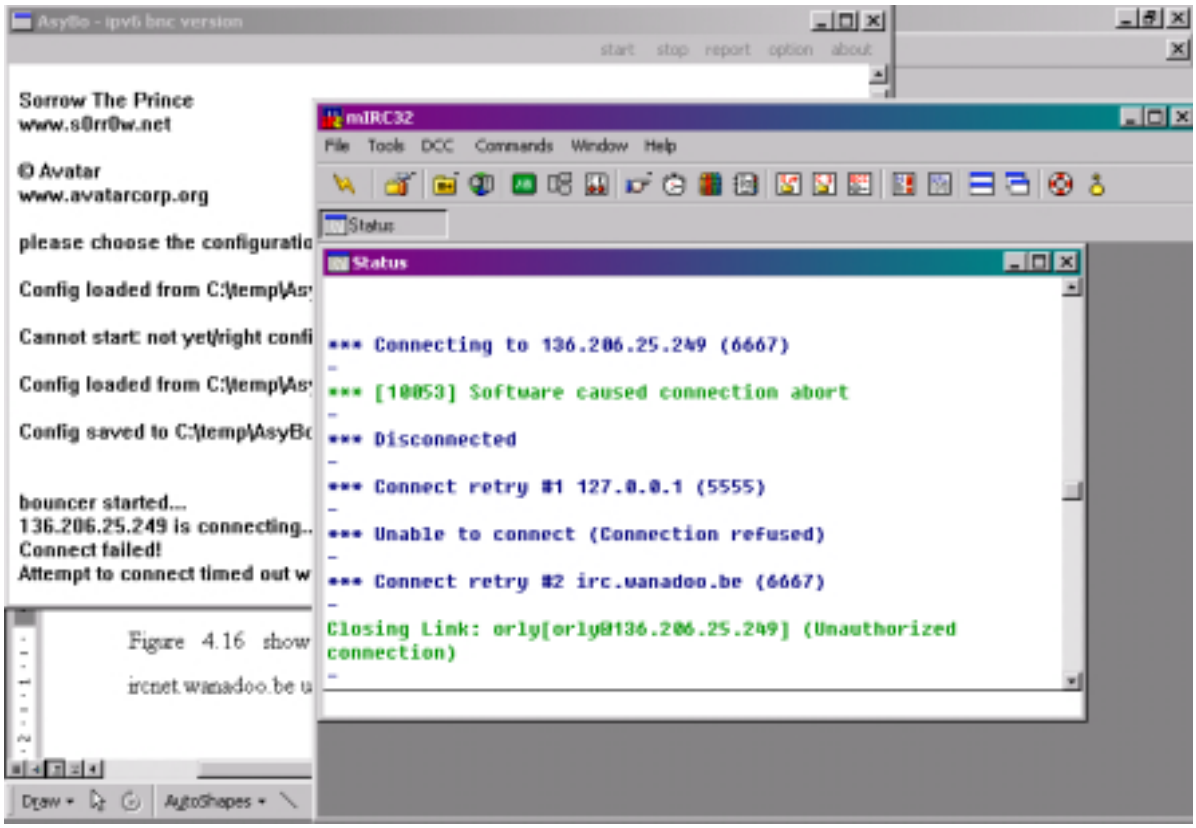


Figure 4.16 Screen shot of MIRC attempting to connect to ircnet.wanadoo.be using the bouncer

In this particular case, the connection the bouncer was trying to make timed out before a connection could be made to the server – probably because the server was down at the time. MIRC then tries to connect to other IRC channels that have previously been specified; localhost (127.0.0.1) was another of the attempts to get the bouncer to work, and irc.wanadoo.be is another of the IRC servers available[19].

When there is a more solid conversion of IRC clients and servers to IPv6, and more documentation and information is available on how to set up the servers and how to connect to the clients, IRC will become a good test for the robustness of client-servers on the Internet.

[19] This only takes connections from subscribed users to wanadoo in Belgium, over IPv4, so I'm not eligible to connect.

4.3 Benchmarking

There are a number of different benchmarking software packages available for FreeBSD that provide statistics for network and system performance. Netperf, the network performance evaluation tool is one of the most frequently used packages. It works using two programs: *netserver* (the server) and *netperf* (the measurement tool). There are a number of scripts available to give control over a number of test variables such as:

- specification of desired confidence levels for the tests (Netperf will warn the user if these levels were not achieved).
- filling send buffers with specified data (to beat compression schemes)
- specification of send/receive buffer alignments and data offsets
- requesting cpu utilization and service demand calculations
- specification of sizes of data to send

The snapshot_script, which takes the parameter of the hostname or IP address of the machine, and performs a quick "snapshot" of the performance of two nodes, performs the following tests[20]:

- TCP Stream test with 56Kbytes socket buffers and 4Kbyte sends
- TCP Stream test with 32Kbytes socket buffers and 4Kbyte sends
- TCP Request/Response test with 1 byte requests and 1 byte responses
- UDP Request/Response test with 1 byte requests and 1 byte responses
- UDP Stream test with 56Kbytes socket buffers and 4Kbyte sends
- UDP Stream test with 32Kbytes socket buffers and 4Kbyte sends

Netperf is supposed to have been patched to allow for IPv6 addresses, according to `http://www.FreeBSD.org`, but the output from my test, shown in figure 4.17 below, doesn't give this impression.

[20] Netperf was obtained using the ports tree on FreeBSD, and the information given above is documented comments in the script 'snapshot_script'

```
-------------------------------------
Testing with the following command line:
/usr/local/netperf/netperf -t TCP_STREAM -l 60 -H 3ffe:b00:c18:1fff::21f -i 10,3 -
I 99,5 -- -s 57344 -S 57344 -m 4096

establish_control: could not resolve the destination 3ffe:b00:c18:1fff::21f



-------------------------------------
Testing with the following command line:
/usr/local/netperf/netperf -t TCP_STREAM -l 60 -H 3ffe:b00:c18:1fff::21f -i 10,3 -
I 99,5 -- -s 32768 -S 32768 -m 4096

establish_control: could not resolve the destination 3ffe:b00:c18:1fff::21f



-------------------------------------
Testing with the following command line:
/usr/local/netperf/netperf -t TCP_RR -l 60 -H 3ffe:b00:c18:1fff::21f -i 10,3 -I
99,5 -- -r 1,1

establish_control: could not resolve the destination 3ffe:b00:c18:1fff::21f



-------------------------------------
Testing with the following command line:
/usr/local/netperf/netperf -t UDP_RR -l 60 -H 3ffe:b00:c18:1fff::21f -i 10,3 -I
99,5 -- -r 1,1

establish_control: could not resolve the destination 3ffe:b00:c18:1fff::21f



-------------------------------------
Testing with the following command line:
/usr/local/netperf/netperf -t UDP_RR -l 60 -H 3ffe:b00:c18:1fff::21f -i 10,3 -I
99,5 -- -r 516,4

establish_control: could not resolve the destination 3ffe:b00:c18:1fff::21f

-------------------------------------
Testing with the following command line:
/usr/local/netperf/netperf -t UDP_STREAM -l 60 -H 3ffe:b00:c18:1fff::21f -i 10,3 -
I 99,5 -- -s 32768 -S 32768 -m 4096

establish_control: could not resolve the destination 3ffe:b00:c18:1fff::21f



-------------------------------------
Testing with the following command line:
/usr/local/netperf/netperf -t UDP_STREAM -l 60 -H 3ffe:b00:c18:1fff::21f -i 10,3 -
I 99,5 -- -s 32768 -S 32768 -m 1024

establish_control: could not resolve the destination 3ffe:b00:c18:1fff::21f
```

Figure 4.17 Output of **netperf** command on FreeBSD machine

Putting the address in square brackets as with Internet Explorer and Tera Term Pro, doesn't work either because the shell script doesn't recognise this format. It can only be concluded that netperf isn't quite ready to deal with IPv6 yet.

Network benchmarking software for Windows 2000 was difficult to obtain. It was later discovered that Windows 2000 Server, the other version of Windows 2000, had benchmarking software as part of its standard installation. When this was discovered, it was too late to install another Operating System and set up the protocol and the tunnel all over again.

Chapter 5

Routing and Domain Name Service (DNS)

Chapter 2 discussed at great lengths the changes that have been made to the IPv6 Protocol, but the protocols and services dependent on IP (such as DNS, DHCP, OSPF etc.) will operate almost exactly as they do for IPv4. The only major changes will be to accommodate the larger addressing format.

5.1 Routing IPv6

Routing IPv6 will require either a stand-alone IPv6 router or a dual-stack router to forward IPv6 packets. Routers running both versions of the protocol will be administered in much the same way as they are now. The development of IPv6 versions of routing protocols such as Open Shortest Path First (OSPF), Routing Information Protocol (RIP) and Border Gateway Protocol (BGP) has been happening for a while now[21].

Even if both protocols are running on the one router/host, it's expected that they will be administered separately. Though it is also possible to align the two protocols by using similar domain boundaries and subnets. There are advantages to both approaches; a separate architecture would remove the seemingly random addressing structure of a lot of IPv4 networks, and an independent IPv6 implementation allows for a clean slate to create a proper hierarchal network topology, and will allow for the connection to more than one ISP. Both offer the prospect of starting afresh with the hope of avoiding today's routing issues such as route aggregation, and the inefficient numbering of networks. The topic of routing

---

[21] The GNU zebra routing software supports the following routing protocols, on a number of UNIX platforms:

bgpd	Manages BGP-4 and BGP-4+ protocol
ripd	Manages RIPv1, v2 protocol
ripngd	Manages RIPng protocol
ospfd	Manages OSPFv2 protocol
ospf6d	Manages OSPFv3 protocol
zebra	for Kernel routing table update and routing information redistribution between above protocols

http://www.zebra.org/

protocols is a very extensive one, so this will only be an overview of the changes to the current protocols, for their IPv6 compatible versions.

OSPF6 is the third version of the protocol, and it supports IPv6 [15]. OSPF is the IETFs standard Interior Gateway Protocol (IGP). There have only been minor changes from OSPFv2, which supports Ipv4, mainly to accommodate the increased address space. It still uses a Link-State Algorithm (now it has 128-bit link-state records instead of 32-bit), but now it handles protocol processing on a 'per-link' basis now, as opposed to its previous handling of it on a 'per-subnet' basis. Also, in OSPF6, packet authentication has been removed, as it is now handled by IPv6s Authentication Payload option header [16].

The fundamental mechanisms of OSPF: hello, flooding, DR (designated router) election, area support, Shortest Path First calculations, etc., all remain the same. Again it is expected that OSPF6 routers will run in parallel with the database for IPv4 network topologies, and the two versions will operate in the same manner as machines with IPv4 and IPv6 on them. Because there are very few fundamental changes being made to OSPF, it is expected that Network Engineers who are already experienced in using OSPF will experience little or no problems adapting to the new version [5].

A new version of Routing Information Protocol (RIP), called RIPng (next generation) has also been developed [17]. RIP is also an IGP, and is generally used for moderately sized Autonomous Systems (AS). RIPng is similar to its predecessor RIPv2 for IPv4 in that it's still based on the Bellman-Ford (distance vector) algorithm, and it is still limited to a metric limit of 15 (i.e. the networks longest path can't be greater than 15 hops). RIPng has also been modified to handle the new 128-bit addresses and new address syntax, in it's routing table.

New versions of Exterior Gateway Protocols (EGP), such as Border Gateway Protocol (BGP) have to be developed for IPv6. Exterior gateway protocols are used to establish connections across the Internet Backbone, between large corporations, academic and research networks, and other large autonomous systems. BGP is used today, to route the majority of packets across the Internet, so it is well known to Network Engineers and providers. Unfortunately, it also was unable to deal with the mass expansion of the Internet, and it has been decided by the Internet community to develop the Internet Domain Routing Protocol (IDRP) for IPv6, in favour of trying to develop BGP for routing between

autonomous systems for IPv6[22]. IRDP was developed for the exchange of Connectionless Network Layer Protocol (CNLP) packets, by inter-domain routing.

…The Inter-Domain Routing Working Group is chartered to standardize and promote the Border Gateway Protocol Version 4 (BGP-4) and ISO Inter- Domain Routing Protocol (IDRP) as scalable inter-autonomous system routing protocols capable of supporting policy based routing for TCP/IP internets. The objective is to promote the use of BGP-4 to support IP version 4 (IPv4). IDRP is seen as a protocol that will support IPv4 as well as the next generation of IP (IPv6) [18].

5.2 DNS for IPv6

As with the routing protocols described in the last section, the changes to DNS for IPv6 are small; it functions much the same as it does for IPv4; DNS is used to map hostnames to IP addresses. Again, it's the new address format and size that affect it. The IETF devised a new standard, which specifies the new 128-bit DNS record type named "AAAA" (quad A) that maps domain names to an IPv6 address in the same way that the "A" DNS record type matches domain names to an IPv4 address [19]. Since that was proposed in 1995, a new RFC [20] has been written to amend this, which describes the "A6" record type, and has support for the earlier 'quad A' proposal, that has already been implemented in places. Also from this RFC, are the goals for the storage of IPv6 addresses as follows:

- The new resource type, "A6", is defined to map a domain name to an IPv6 address, with a provision for indirection for leading "prefix" bits.
- Existing queries that perform additional section processing to locate IPv4 addresses are being refined so as to process for both versions of the protocol.
- A new domain IP6.ARPA is to be established for the purpose of providing 'reverse-lookups' based on an IPv6 address (i.e. if you do a reverse lookup for the address 3ffe:700:20:4:200:f8ff:fe75:6bc7, you should get back its hostname of www.6tap.net).
- A new prefix-delegation method is defined based on new DNS features such as [DNAME, BITLBL].

[22] There are no RFCs for IPv6 enabled BGP, where as there are for OSPF, RIP and IDRP

The A6 record type is specific to the Internet (IN) class and has type number 38, the same as the AAAA record type. It takes the format

Figure 5.1 Resource Data (RDATA) portion of the A6 record.

where the 'Prefix Length' is encoded as an 8-bit integer whose value is between 0 and 128 inclusive. The 'Address Suffix' is encoded in network order (that is, highest-order octet first), and there must be *exactly* enough octets to fill this field, and it must contain a number of bits equal to 128 minus the 'prefix length' (i.e. if the prefix is 128-bits long, then there is no address suffix component). It is suggested, that an A6 record that is only going to be used as a prefix for other A6 records should have all the insignificant trailing bits set to zero. Also, the name of the prefix, encoded as a domain name, must not be compressed.

The new method of performing reverse lookups relies on a new DNS label type [BITLBL], the "bit string label" type. It represents an arbitrary string of bits, which is treated as a hierarchal sequence of one-bit domain labels.

Once an IPv6-capable DNS is in place, dual-stack hosts can interact interchangeably with IPv6 and IPv4 nodes. When a query for an IP address is received, and the DNS locates an A6/AAAA record that holds an IPv6 address, or an A record that holds an IPv4 address, the resolver library can either *filter*, or *order* the results to be returned to the application. Thus it can influence which version of IP is used to communicate with that node. For filtering, the resolver library can either

- Return an IPv6 address to the application,
- Return an IPv4 address to the application,
- Return both types of addresses to the application.

So, if it returns an IPv6 address, the application will communicate with the node using IPv6; if it returns an IPv4 address, the application will deal with the node using IPv4. If it returns both, then the application has the choice of which address to use (and therefore which version of the protocol).

In the case where both types of addresses are returned, the resolver library can elect the order in which the addresses are returned (i.e. IPv6 first, or IPv4 first). Since most applications will try the address in the order they are returned, this allows the application to have an IP 'preference'. The decision to filter or order results from queries is

implementation specific; the most commonly used DNS implementation is Berkeley Internet Name Domain (BIND), and its latest version, version 9.1.1 has support for IPv6[23] [21].

Fundamentally, the domain name service for IPv6 works in the same manner as it does for IPv4, so administrators who need to establish IPv6 DNS should not experience too much difficulty implementing the changes.

[23] IP version 6 support in BIND - Answers DNS queries on IPv6 sockets, can handle IPv6 resource records (A6, DNAME, etc.) and Bitstring Labels, and has an experimental IPv6 Resolver Library.

Chapter 6

Conclusions

The transition from Internet Protocol version four (IPv4), to Internet Protocol version six (IPv6), will be wholly dependent on the speed that the current Internet Protocol dependent protocols, and applications are updated to cope with IPv6. From the work covered during this project, it can be seen that there is already a solid move towards updating the Internet Protocol dependent protocols, such as Domain Name Service (DNS), and the various routing protocols, such as (Route Information Protocol next generation) RIPng, and Open Shortest Path First (OSPF). There has also been considerable work done in updating Internet Protocol dependent applications such as Telnet, Secure Shell (SSH), and web browsers.

The next generation Internet technologies are being investigated as a means to overcome the limitations faced today with IPv4. One effort will evaluate a new implementation of Transmission Control Protocol (TCP), that is aimed at supporting constant-bit-rate traffic such as video streams. IPv6 is an answer to the long-term Internet addressing limitations of IPv4. There are a number of research projects underway at present which will make practical use of IPv6s Quality of Service, such as Telemicroscopy, Video Streaming of conferences and meetings, and on a slightly lighter note, networked sreamed Quake[24]. The development of Internet 2 (located at the StarTAP in Chicago), and 6Tap, it's IPv6 network, allows for the rapid development of these major scientific developments, and the wide scale distribution of the findings. This will help demonstrate the validity of this next generation Internet technology.

The setting up and administrating of two machines posed a couple of problems at first, due to hardware incompatibility, but once these problems were ammended, the two test machines ran smoothly without any faults (even when both of the Operating Systems were upgraded to later versions). The configuration of the tunnels for the Windows 2000 machine, and the FreeBSD machine were easily done with the help of a configuration batch file and perl script respectively from the tunnel provider, but the understanding of how the tunnels would work and how Internet Protocol version six worked required a lot of

[24] http://www.viagenie.qc.ca/en/ipv6/quake/ipv6-quake.shtml

perliminary research. Chapter 2 of this report, describes in detail the fundamental changes that have been made to IPv6, and how these changes affect the usage of the protocol.

Once the addressing structure and the address synax was understood, the understanding of the header changes and the affects they would have all fell in to place. This project has allowed for an indepth understanding of Internet Protocol to be achieved, as well as an indepth knowledge of the protocols which are dependent on it. When this project was started, the author only had a good working knowledge of how IPv4 and TCP/IP worked. Now an all round solid knowledge of Networking the Internet has been gained.

The setting up of a sole IPv6 network, with one machine acting as a gateway to the IPv4 Internet, and using a tunnel, to the IPv6 6Bone, was something that was hoped to be achieved, but due to time limitations for this project it could not. The establishment of an IPv6 router, and IPv6 routed traffic analysis is suggested as a future area of work in this field. The establishment of an IPv6 Domain Name Server (DNS) would also be recommended for further work in this field.

"In protocol design, perfection has been reached not when there is nothing left to add, but when there is nothing left to take away." -- Ross Callon

Bibliography

1. Bay Networks *"White Paper on Ipv6"* - 1997
   http://www.baynetworks.com/Products/Routers/Protocols/2789.html
   (I have noticed since I printed out a copy of this document in September, that Bay Networks was acquired by Norton Networks, and this URL is no longer valid, but I have found a copy of the document at:
   http://www.cs-ipv6.lancs.ac.uk/ipv6/documents/papers/BayNetworks/.)
2. Microsoft Windows 2000 Server *"Introduction to IP version 6 - White Paper"*
   http://www.microsoft.com/windows2000/library/howitworks/communications/nameadrmgmt/introipv6.asp

References

[1] *"The history of the Internet" By Dave Kristula,* March 1997
http://www.davesite.com/webstation/net-history.shtml

[2] Bay Networks *"White Paper on Ipv6, Part One: The Business case for IPv6"* - 1997
http://www.baynetworks.com/Products/Routers/Protocols/2789.html

[3] *"TCP/IP limitations undone"* By Bob Melford, December 17 2000
http://www.unixinsider.com/swol-01-1997/swol-01-ipv6.html

[4] Microsoft Windows 2000 Server *"Introduction to IP version 6 - White Paper"*
http://www.microsoft.com/windows2000/library/howitworks/communications/nameadrmgmt/introipv6.asp

[5] Bay Networks *"White Paper on Ipv6, Part Two: The Technical case for IPv6"* - 1997
http://www.baynetworks.com/Products/Routers/Protocols/2789.html

[6] *"Internet Protocols and Economics"* By Liam Bedford, May 1998

[7] *"IPv6: The Next Generation Internet"* By Dan Knight
http://www.lowendmac.com/tech/ipv6.shtml

[8] RFC 2373 *"IP version 6 Addressing Architecture"*, July 1998
http://www.landfield.com/rfcs/rfc2373.html

[9] *"The migration from IPv4 to IPv6"* By Silvano Gai, August 2000
http://www.ip6.com/us/paper/migr/migr.htm

[10] "Getting Started with the Microsoft IPv6 Technology Preview for Windows 2000", December 2000
http://msdn.microsoft.com/downloads/sdks/platform/tpipv6/start.asp

[11] *"FreeBSD Hypertext Man Pages"*, April 10th 1999
http://www.freebsd.org/cgi/man.cgi?query=faith&sektion=4&apropos=0&manpath=FreeBSD+4.2-RELEASE

[12] http://www.vermicelli.pasta.cs.uit.no/ipv6/software.html

[13] http://win6.goto.info.waseda.ac.jp/TeraTerm/index.html

[14] http://unfix.org/projects/ipv6/

[15] RFC 2740 *"OSPF for IPv6"*, December 1999
http://www.faqs.org/rfcs/rfc2740.html

[16] *"IPv6: The Next Generation TCP/IP Network Protocol"* By David A. Ranch
http://www.ecst.csuchico.edu/~dranch/NETWORKS/IPV6/tsld020.htm

[17] RFC 2080 *"RIPng for IPv6"*, January 1997.
http://www.faqs.org/rfcs/rfc2080.html

[18] *"Inter-Domain Routing (IDR) Charter"* - From the 30th IETF Meeting, July 1994.
http://www.ietf.cnri.reston.va.us/proceedings/94jul/charters/idr- charter.html

[19] RFC 1886 *"DNS Extensions to Support IP Version 6"* December 1995
http://www.faqs.org/rfcs/rfc1886.html

[20] RFC 2874 *"DNS Extensions to Support IPv6 Address Aggregation and Renumbering"*, July 2000
http://www.landfield.com/rfcs/rfc2874.html

[21] Internet Software Consortium Bind 9.1.1
http://www.isc.org/products/BIND/bind9.html

Appendix A

Rate of growth of networks and IPv4 address space occupancy[25]

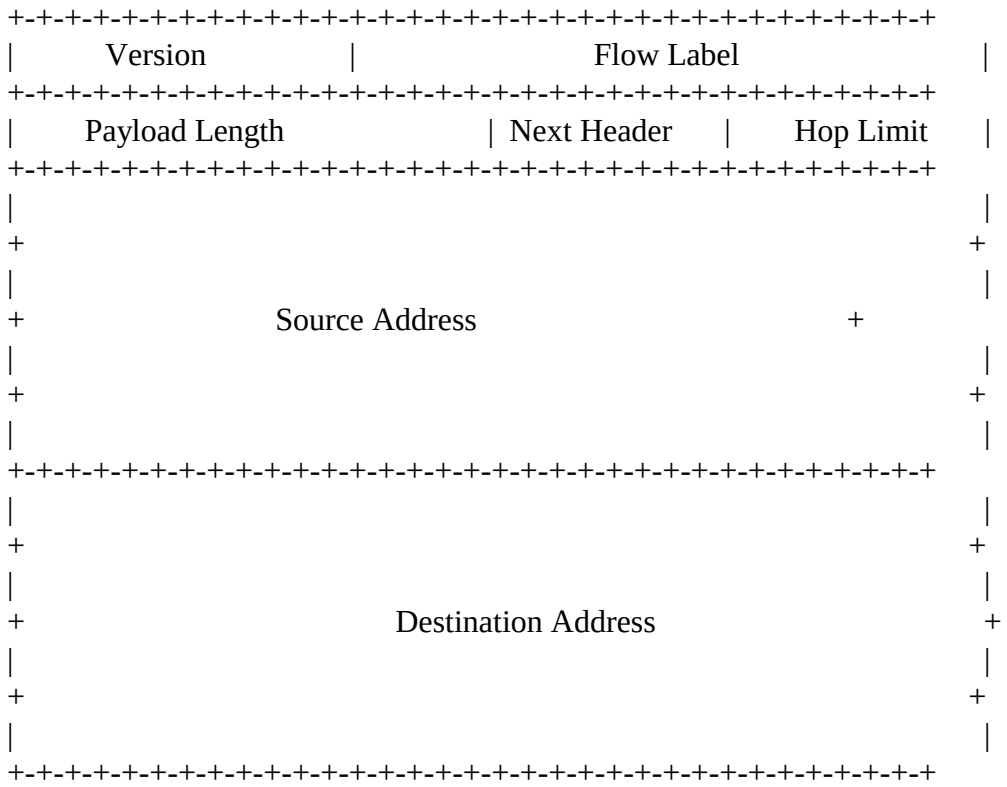
Date	Host	Networks of class:		
		A	B	C
Jan 97	16,146,000			
Jun 96	12,881,000			
Jan 96	9,472,000	92	5,655	87,924
Jul 95	6,642,000	91	5,390	56,057
Jan 95	4,852,000	91	4,979	34,340
Oct 94	3,864,000	93	4,831	32,098
Jul 94	3,212,000	89	4,493	20,268
Jan 94	2,217,000	74	4,043	16,422
Oct 93	2,056,000	69	3,849	12,615
Jul 93	1,776,000	67	3,728	9,972
Apr 93	1,486,000	58	3,409	6,255
Jan 93	1,313,000	54	3,206	4,998

[25] http://www.ip6.com/us/paper/intro/introduction.htm

Appendix B

IPv6 Header Format[26]

The IPv6 header, although longer than the IPv4 header, is considerably simplified. A number of functions that were in the IPv4 header have been relocated in extension headers or dropped.



- Version - Internet Protocol version number. IPng has been assigned version number 6. (4-bit field)

- Flow Label - This field may be used by a host to label those packets for which it is requesting special handling by routers within a network, such as non-default quality of service or "realtime" service. (28-bit field)

[26] The Recommendation for the IP Next Generation Protocol, January 1995 http://www.faqs.org/rfcs/rfc1752.html

- Payload Length - Length of the remainder of the packet following the IPv6 header, in octets. To permit payloads of greater than 64K bytes, if the value in this field is 0 the actual packet length will be found in a Hop-by-Hop option. (16-bit unsigned integer)

- Next Header - Identifies the type of header immediately following the IPv6 header. The Next Header field uses the same values as the IPv4 Protocol field (8-bit selector field)

- Hop Limit - Used to limit the impact of routing loops. The Hop Limit field is decremented by 1 by each node that forwards the packet. The packet is discarded if Hop Limit is decremented to zero. (8-bit unsigned integer)

- Source Address - An address of the initial sender of the packet. (128-bit field)

- Destination Address - An address of the intended recipient of the packet (possibly not the ultimate recipient, if an optional Routing Header is present). (128-bit field)

Appendix C

Differences between IPv4 and IPv6[27]

IPv4	IPv6
Source and Destination addresses are 32-bits (4-Bytes) in length.	Source and Destination addresses are 128-bits (16-Bytes) in length.
IPSec support is optional.	IPSec support is required.
No identification of payload for QoS handling by routers is present within the IPv4 header.	Payload identification for QoS handling by routers is included in the IPv6 header using the Flow Label field.
Fragmentation is supported at both routers and the sending host.	Fragmentation is not supported at routers. It is only supported at the sending host.
Header includes a checksum.	Header does not include a checksum
Header includes options.	All optional data is moved to the IPv6 extension headers.
Address resolution protocol (ARP) uses broadcast ARP request frames to resolve an IPv4 address to a link layer address.	ARP Request frames are replaced with multicast Neighbour solicitation messages.
Internet Group Management Protocol (IGMP) is used to manage local subnet group membership.	IGMP is replaced by Multicast Listener Discovery (MLD) messages.
ICMP Router Discovery is used to determine the IPv4 address of the best default gateway and is optional.	ICMPv4 Router Discovery is replaced by ICMPv6 Router Solicitation and Router Advertisement messages and is required.
Broadcast addresses are used to send messages to all nodes on a subnet.	There are no IPv6 broadcast addresses. Instead, a link-local scope all-nodes multicast address is used
Must be configured either manually or through DHCP.	Does not require manual configuration or DCHP (incorrect – has Autoconfiguration of

[27] Microsoft Windows 2000 Server *"Introduction to IP version 6 - White Paper"*
http://www.microsoft.com/windows2000/library/howitworks/communications/nameadrmgmt/introipv6.asp

	addresses, that can uses DHCP server)
Uses host address (A) resource records in the Domain Name System (DNS) to map host names to IPv4 addresses.	Uses host address (A6) and (AAAA) resource records in the Domain Name System (DNS) to map host names to IPv6 addresses.
Uses pointer (PTR) resource records in the IN-ADDR.ARPA DNS domain to map IPv4 addresses to host names	Uses pointer (PTR) resource records in the IP6.ARPA and IP6.INT DNS domain to map IPv6 addresses to host names respectively.

Appendix D

Current Allocation of the IPv6 Address Space[28]

IPv4 Address	IPv6 Address
Internet Address Classes	Not implemented in IPv6
Multicast Addresses (244.0.0.0/4)	IPv6 Multicast Addresses (FF00::/8)
Broadcast Addresses	Not implemented in IPv6
Unspecified Address is 0.0.0.0	Unspecified Address is ::
Loopback address is 127.0.0.1	Loopback address is ::1
Public IP addresses	Aggregatable global unicast addresses
Private IP addresses (10.0.0.0/8, 172.16.0.0/12, and 192.168.0.0/16)	Site-local addresses (FEC0::/48)
Autoconfigured addresses (169.254.0.0/16)	Link-local addresses (FE80::/64)
Text representation: Dotted decimal notation	Text representation: Colon hexadecimal format with suppression of leading zeros and zero compression. IPv4-compatible addresses are expressed in dotted decimal notation.
Network bits representation: Subnet mask in dotted decimal notation or prefix length	Network bits representation: Prefix length notation only.
DNS name resolution: IPv4 host address (A) resource record	DNS name resolution: IPv6 host address (A6) and (AAAA) resource record.
DNS reverse resolution: IN-ADDR.ARPA domain	DNS reverse resolution: IP6.ARPA and IP6.INT domain

[28] Microsoft Windows 2000 Server *"Introduction to IP version 6 - White Paper"*
http://www.microsoft.com/windows2000/library/howitworks/communications/nameadrmgmt/introipv6.asp

Appendix E

Perl script provided by Freenet6[29], to set up IPv6 addresses on the FreeBSD machine to allow me to tunnel to the 6Bone.

```
#!/usr/bin/perl
# Perl script for autotunnel IPv6 with Freenet6.net
  print "Your system is using FreeBSD with KAME stack\n";
# Find an available interface for tunnelling , so flags 8010 = available if
  print "This script is finding an available interface for the tunnel\n";
  system(`ifconfig -a | grep gif | grep 8010 > /tmp/abc`);
 open(KAME,"/tmp/abc");
 @lines=<KAME>;
  chop (@lines);

# Take the first interface gif available in the list
  ($if,$data)=split (':',$lines[0]);
  print "The available interface for the tunnel is : $if\n";
  close(KAME);

# Somes informations about tunnels values
  print "This script will create a tunnel between this computer\n";
  print "and the Freenet6 server (tunnels server)\n";
  print "Your IPv6 address (your tunnel end point) is 3ffe:b00:c18:1fff:0:0:0:21f  \n";
  print "We establish a tunnel to the Freenet6 server at 3ffe:b00:c18:1fff:0:0:0:21e \n";
  print "Your IPv4 address is : 136.206.25.248 \n";
  print "The IPv4 address of the Freenet6 server is : 206.123.31.102 \n";

 # Setup the tunnel with values from Freenet6
      system(`gifconfig $if 136.206.25.248 206.123.31.102`);
      system(`ifconfig $if inet6 3ffe:b00:c18:1fff:0:0:0:21f
              3ffe:b00:c18:1fff:0:0:0:21e prefixlen 128 alias`);
       system(`ifconfig $if up`);
       system(`route add -inet6 default 3ffe:b00:c18:1fff:0:0:0:21f`);

      print "End of the script for IPv6 with KAME \n";
```

[29] http://www.freenet6.net

Appendix F

Batch file provided by Freenet6[30], to set up IPv6 addresses on the Windows 2000 machine to allow me to tunnel to the 6Bone. I had to use their Windows NT4.0 set up version as they don't provide a Windows 2000 set up, but it works exactly the same, because it uses the same research TCPIPv6 stack.

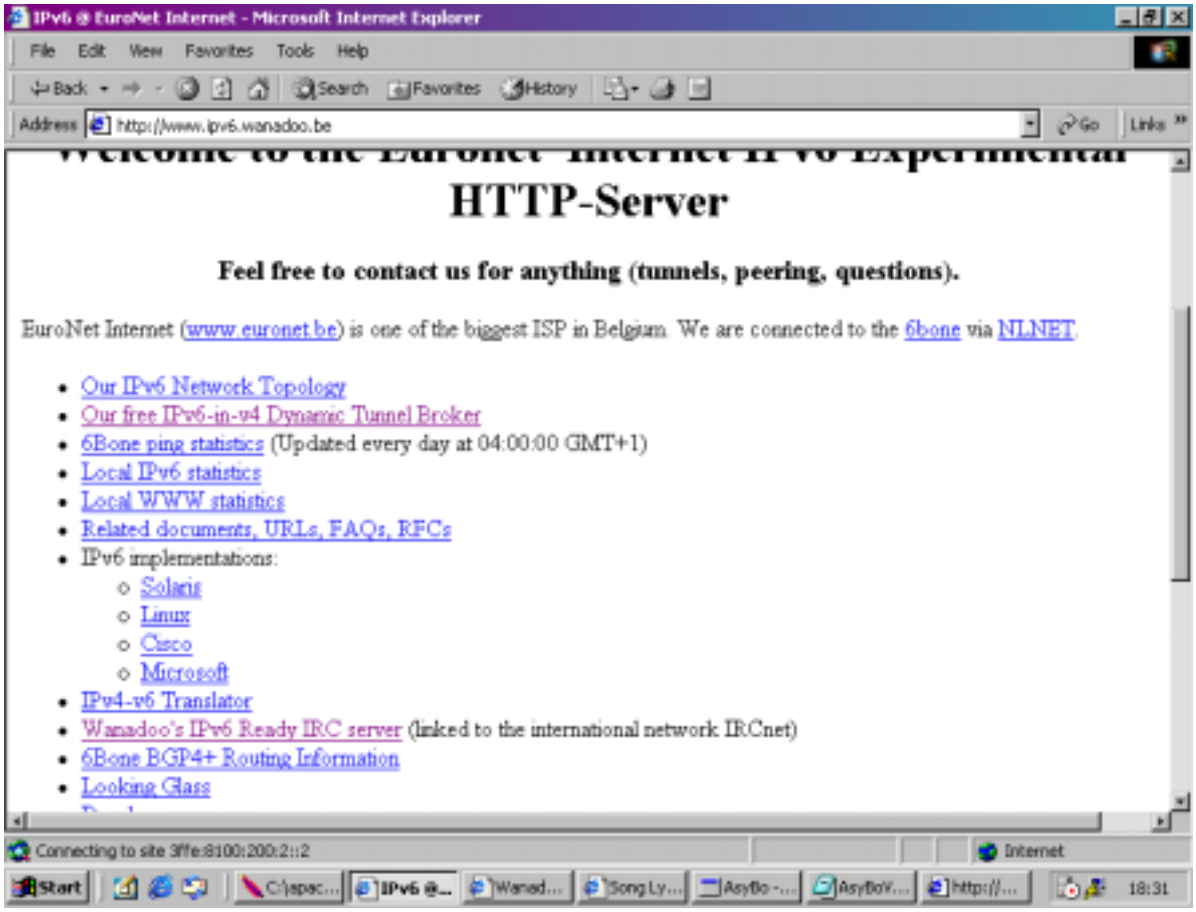
```
@echo off
Rem ************************************
Rem * IPv6 tunnel config under NT 4.0   *
Rem * Freenet6.net                          *
Rem * IPv6 stack for MSRipv6 version 1.2*
Rem ************************************
Rem * For NT 4.0 version 0.2
Rem * We suppose you copied ../ipv6kit/*.exe to %windir%/system32
Rem We keep a log
date /T >> tunnel.log
time /T >> tunnel.log

Rem ***** Script for NT to setup an IPv6 tunnel "on-line" **********

:begin
ipv6.exe rtu ::/0 2/::206.123.31.102 pub >> tunnel.log
if errorlevel 1 goto fail1
echo The IPv4 address tunnel server is correctly configured
goto second

:second
ipv6.exe adu 2/3ffe:b00:c18:1fff:0:0:0:193 >> tunnel.log
if errorlevel 1 goto fail2
echo The IPv6 address tunnel client is correctly configured
goto success

goto end

:faill
Echo Impossible to enter the IPv4 address for the Freenet6 server
Echo Impossible to enter the IPv4 address for the Freenet6 server >> tunnel.log
goto end

:fail2
Echo Impossible to enter the IPv6 address for the client (your IPv6 address)
Echo Impossible to enter the IPv4 address for the client (your IPv6 address) >> tunnel.log
goto end

:success
```

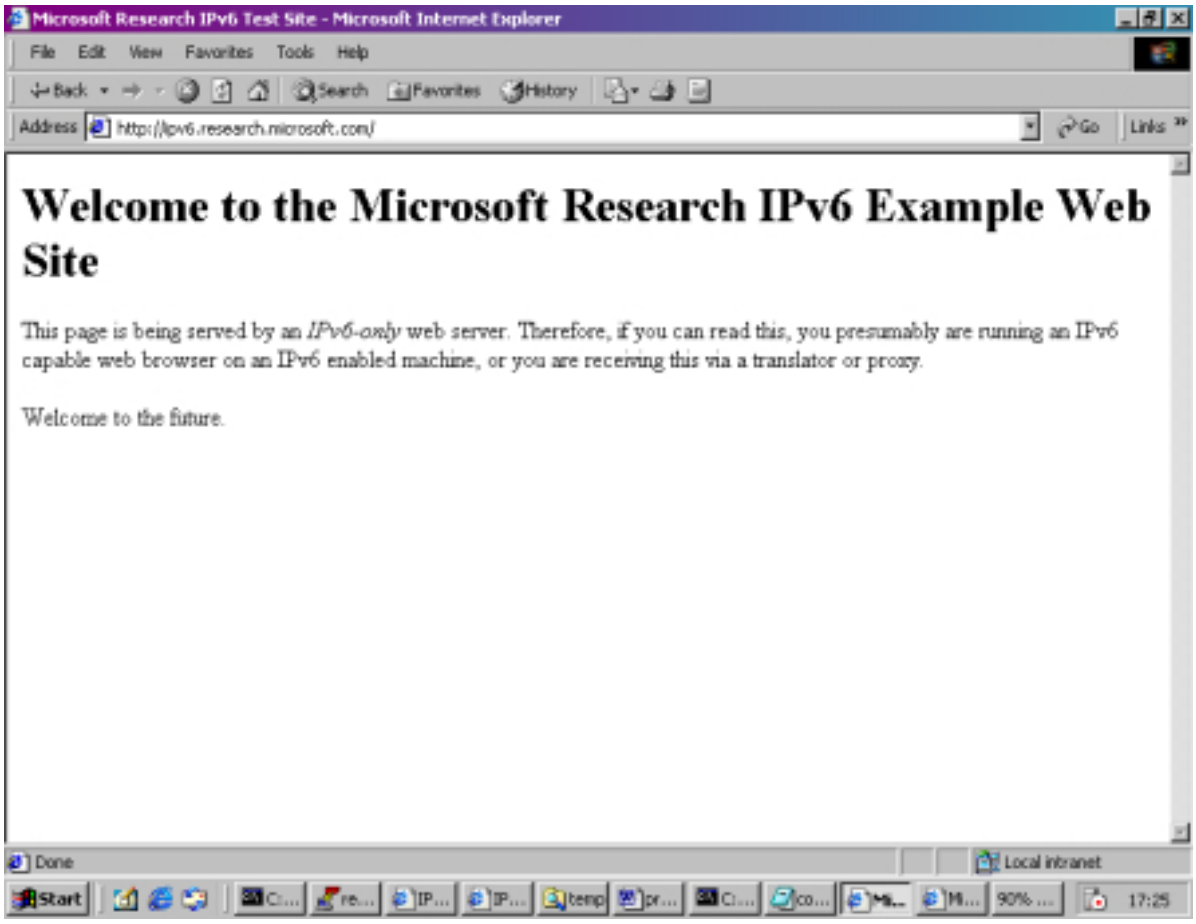
[30] http://www.freenet6.net

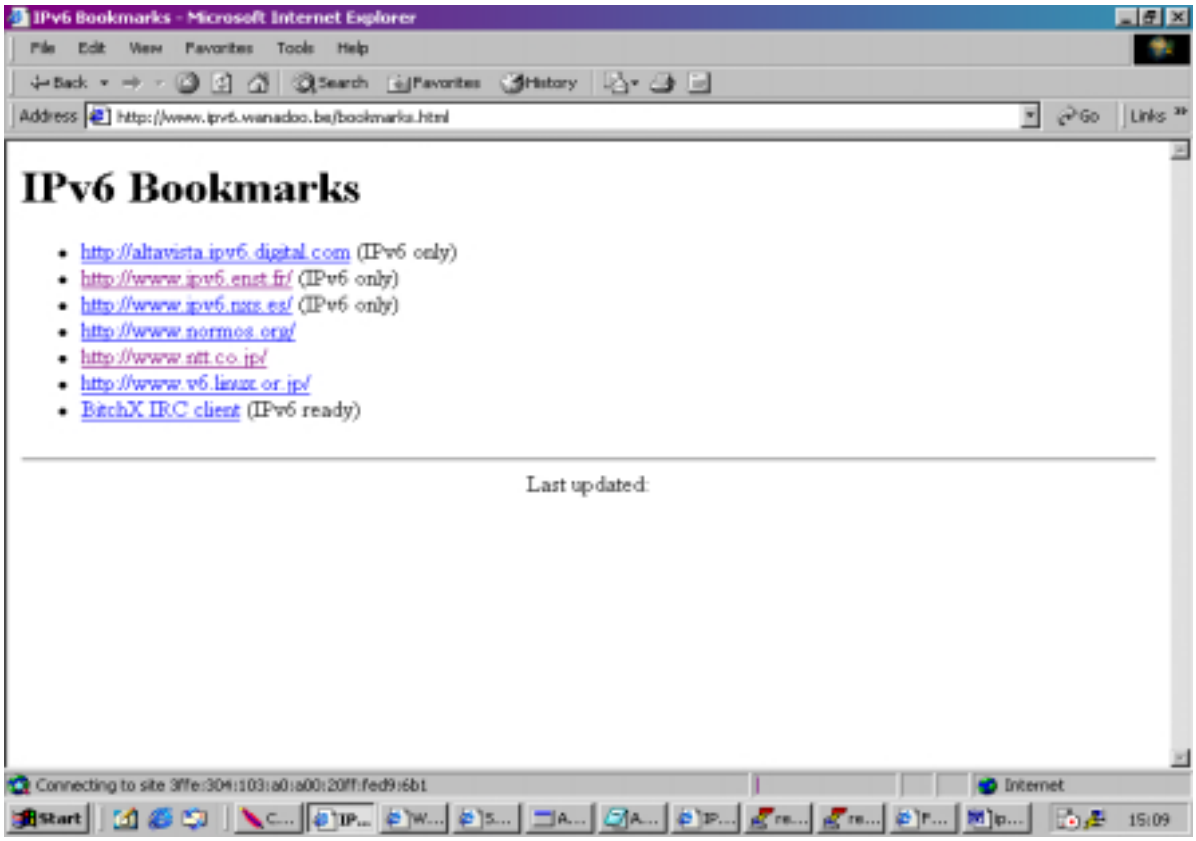
```
Echo Now, you're ready to use IPv6 connectivity with the 6Bone
Echo Your computer is configured with the IPv6 address 3ffe:b00:c18:1fff:0:0:0:193
Echo The server is configured with the IPv6 address 3ffe:b00:c18:1fff:0:0:0:192
Echo Your computer has the IPv4 address 136.206.25.249
Echo The server has the IPv4 address 206.123.31.102
Echo Your computer has a connectivity with the 6Bone !!!
goto end

:end

Echo End of the script
```

Appendix G

The following are some screen shots of IPv6 specific web sites on the web. In the location bar, the name of the web server I want to connect to is shown. I can only connect to this site because the machine has IPv6 running on it, and the browser can then resolve the domain name to its IPv6 address. The status bar down the bottom of the browser shows this address as the page is being downloaded.




Microsoft Research IPv6 Test Site - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://ipv6.research.microsoft.com/
Go Links
Welcome to the Microsoft Research IPv6 Example Web Site
This page is being served by an IPv6-only web server. Therefore, if you can read this, you presumably are running an IPv6 capable web browser on an IPv6 enabled machine, or you are receiving this via a translator or proxy.
Welcome to the future.
Done
Local intranet
Start
17:25


IPv6 Bookmarks - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://www.ipv6.wanadoo.be/bookmarks.html
Go Links
IPv6 Bookmarks
http://altavista.ipv6.digital.com (IPv6 only)
http://www.ipv6.enst.fr/ (IPv6 only)
http://www.ipv6.rgtc.es/ (IPv6 only)
http://www.normos.org/
http://www.ntt.co.jp/
http://www.v6.linux.or.jp/
BitchX IRC client (IPv6 ready)
Last updated:
Connecting to site 3ffe:304:103:a0:a00:20ff:fed9:6b1
Internet
Start
15:09