



Experimental Deployment of a Web Server  
over the Ipv6 Backbone

by

Mr. James Clark Baldwin

A Final Report of the Six-Credit Course  
CS 6998 - CS 6999 System Development Project

Submitted in Partial Fulfillment  
of the Requirements for the Degree of  
Master of Science  
in Computer Information Systems  
Assumption University

March 2004



**Experimental Deployment of a Web Server  
over the Ipv6 Backbone**

by  
Mr. James Clark Baldwin

A Final Report for the Six-Credit Course  
CS6998-CS6999 System Development Project

Submitted in Partial Fulfillment  
of the Requirements for the Degree of  
Master of Science  
in Computer Information Systems  
Assumption University

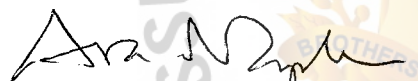
March 2004

Project Title        Experimental Deployment of a Web Server over the IPv6 Backbone  
Name                James Clark Baldwin  
Project Advisor     Dr. Aran Namphol  
Academic Year     March 2004


---


The Graduate School of Assumption University has approved this final report of the six-credit course, CS6998-CS6999 System Development Project, submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Information Systems.

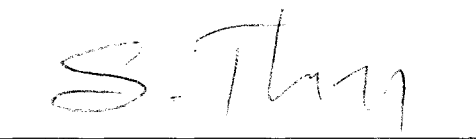
Approval Committee:

  
(Assoc.Prof.Dr. Aran Namphol)  
Advisor

  
(Prof.Dr. Srisakdi Charmonman)  
Chairman

  
(Air Marshal Dr. Chulit Meesajjee)  
Dean and Co-advisor

  
(Asst.Prof.Dr. Vichit Avatchanakorn)  
Member

  
(Assoc.Prof. Somchai Thayarnyong)  
CHE Representative

March 2004

## ABSTRACT

This project report is a two-pronged effort. On the one hand, it details the problems of the IPv4 Internet that led to the decision to revise the Internet protocol and the solutions that are instantiated into IPv6. On the other hand, it details an effort to actually implement a web server over the IPv6 test backbone. As such, it represents sort of a hybrid ‘mini-thesis’ rather than a traditional system development project.

The most extensive section of the report is an extended literature review structured as a detailed comparative analysis of the shortcomings of the IPv4 protocol and the solutions that were devised in IPv6. Topics include IP addressing, headers, extension headers as the version 6 of the Internet Control Message Protocol, Flows and Multicasting, Routing and Autoconfiguration.

This is followed by a chapter dealing with the IETF strategies for the transition from IPv4 to IPv6. Various tunneling mechanisms are described, as is the experimental IPv6 backbone called the 6Bone. Strategies for accessing the IPv6 backbone are presented.

The last section reports on an effort to establish connectivity through the IPv6 network, and, as a proof of concept, to serve HTTP clients through it. Various software solutions are explored, feasibility and cost analyses performed and a candidate matrix constructed. A system based on GNU Linux and the Apache web server was selected. Details of the construction of the working system conclude the report.

## ACKNOWLEDGEMENTS

Exhaustive acknowledgements always carry the danger of offending some valued contributor to one's efforts through inadvertent omission. It therefore seems prudent to avoid mentioning anyone by name, especially since this writer has such a long list of valued mentors, teachers and friends who have contributed in so many ways to this effort.

There are a few people, however, who were so central to the production of this work that it would seem downright churlish not to mention them.

Dr. Aran Namphol, my advisor in this project, taught the first class I attended at Assumption University, the last class I attended and a large number of others in between. He has been a marvelous teacher, a skillful advisor and a treasured friend. I thank him from the bottom of my heart.

The Dean of the Graduate School of Computer Information Systems, Dr. Chulit Meesajjee also deserves special mention. Far from the stereotype of the cold and distant Dean, Dr. Chulit has blessed me with countless warm and fruitful hours in his office and in his classroom. I am deeply grateful for his help, his advice and his friendship.

Domestic happiness obliges me to acknowledge the priceless (and often thankless) role of my wife, Taneeya Runcharoen. She has been an unfailing fountain of support in all things. I could not have done this without her.

To all my other teachers and friends, I am deeply grateful. You have all contributed to me in ways that I shall always treasure. Thank you.

## TABLE OF CONTENTS

<u>Chapter</u>	<u>Page</u>
ABSTRACT	i
ACKNOWLEDGEMENTS	ii
LIST OF FIGURES	vii
LIST OF TABLES	ix
I. INTRODUCTION	1
1.1 Background of the Project	1
1.2 Objectives of the Project	2
1.3 Scope of the Project	3
1.4 Deliverables	4
II. IPV4 PROBLEMS AND IPV6 SOLUTIONS	6
2.1 Overview	6
2.2 The Address Space	7
2.2.1 IPv4 Addresses	7
2.2.2 Stopgap Measures	9
2.2.3 IPv6 Address Size	10
2.2.4 Typographical Expression of IPv6 Addresses	11
2.2.5 Categories of IPv6 Addresses	13
2.2.6 Address Space Assignment	14
2.2.7 Special Addresses	15
2.2.8 Unicast Addresses	16
2.2.9 Anycast Addresses	19
2.2.10 Multicast Addresses	20

<u>Chapter</u>	<u>Page</u>
2.3 The IP Header	22
2.3.1 The IPv4 Header	22
2.3.2 The IPv6 Header	26
2.3.3 IPv6 Extension Headers	32
2.4 ICMPv6 and Other Protocols	38
2.4.1 The Impact of IPv6 on Other Protocols	38
2.4.2 ICMPv6 Overview	39
2.4.3 ICMPv6 Error Reporting Messages	39
2.4.4 ICMPv6 Query Messages	42
2.5 Multicasting and Flows in IPv6	47
2.5.1 IPv6 Multicasting	47
2.5.2 Multicast Routing in IPv6	48
2.5.3 Multimedia and Flows	48
2.5.4 RSVP – The Resource Reservation Protocol	51
2.6 Routing in IPv6	52
2.6.1 Internal Routing Protocols	53
2.6.2 Exterior Routing Protocols	56
2.7 Autoconfiguration	58
2.7.1 Link Local Addresses	59
2.7.2 Stateless Autoconfiguration	59
2.7.3 Stateful Autoconfiguration	61
2.7.4 Site Renumbering	63
2.7.5 Address Resolution	64
2.7.5 Black Hole Detection	64

<u>Chapter</u>	<u>Page</u>
III. THE TRANSITION STRATEGY FOR IPv6	66
3.1 Overview	66
3.2 Dual Stacks	68
3.3 Tunneling	70
3.3.1 Automatic Tunneling	70
3.3.2 Configured Tunneling	71
3.4 Header Translation	73
3.5 The 6Bone	74
3.6 The Tunnel Setup Protocol and Freenet6	77
IV. THE IPv6 WEB SERVER PROJECT	79
4.1 Requirements Analysis	79
4.2 The Existing System	80
4.3 Platform and Software Selection for the Proposed System	82
4.3.1 Hardware for the Proposed System	82
4.3.2 Candidate Operating Systems for the Proposed System	82
4.3.3 Available IPv6 Protocol Stacks	84
4.3.4 HTTP Server Software	85
4.3.5 Candidate System Feasibility Analysis	86
4.4 Design of the Proposed System	92
4.5 Implementation of the System	95
4.5.1 Software Installation on the Server	95
4.5.2 Software Installation on the Client	97
4.6 Testing the System	100
4.6.1 Testing the System on an Ethernet LAN	100



<u>Chapter</u>	<u>Page</u>
4.6.2 Testing the System on the 6Bone IPv6 Backbone	107
V. CONCLUSIONS AND RECOMMENDATIONS	114
5.1 Conclusions	114
5.2 Recommendations	115
APPENDIX A LOCATING THE SOFTWARE PACKAGES	117
APPENDIX B IP PACKET HEADER DUMPS	118
B.1 IPv4 Ping Headers	118
B.2 IPv6 Ping Headers	119
BIBLIOGRAPHY	120



## LIST OF FIGURES

<u>Figure</u>	<u>Page</u>
2.1 IPv6 Hierarchical Global Address Structure	18
2.2 IPv6 Link-local Address Structure	18
2.3 IPv6 Site-local Address Structure	19
2.4 IPv6 Multicast Address Structure	20
2.5 The IPv4 Header	22
2.6 The IPv6 Header	26
2.7 Format of a Source Routing Extension Header	34
2.8 General Format of ICMPv6 Messages	39
2.9 Format of Destination Unreachable Messages	40
2.10 Format of Packet Too Big Messages	41
2.11 Format of Redirection Messages	42
2.12 Format of Echo Request and Reply Messages	43
2.13 Format of Router Solicitation Messages	43
2.14 Format of Router Advertisement Messages	44
2.15 Format of Neighbor Solicitation Messages	45
2.16 Format of Group Membership Query Messages	46
3.1 Dual Stack Schematic	69
3.2 Automatic Tunneling	71
3.3 Configured Tunneling	72
3.4 Header Translation	73
3.5 The 6Bone Backbone at Present	75
4.1 Architecture of the Proposed System	93

<u>Figure</u>	<u>Page</u>
4.2 Server Interface Configuration	101
4.3 Client Interface Configuration	101
4.4 Connectivity Test from Client to Server	102
4.5 Aliases in the etc/hosts File	103
4.6 Ping with Aliased IPv6 Address	104
4.7 Apache Test Page for the Address ipv6-localhost	105
4.8 Apache Test Page for the Address [::1]	106
4.9 Apache Test Page over the Ethernet Network	107
4.10 Initial Freenet6 Connection	108
4.11 The sit1 Interface Added by Freenet6	109
4.12 Result of Freenet6 Configuration during Test Run	110
4.13 First Frame of the KAME Animation	111
4.14 Second Frame of the KAME Animation	111
4.15 Traceroute over IPv6	112
4.16 IPv6 Web Page Served to a Global IPv6 Address	113

## LIST OF TABLES

<u>Table</u>	<u>Page</u>
2.1 Estimated Network Address Populations per Address Size	11
2.2 Type Prefixes for IPv6 Addresses	15
2.3 Interpretation of Scope Values in Multicast Addresses	21
2.4 Priorities for congestion controlled traffic	27
2.5 Next Header Codes	29
4.1 Computer Hardware Specifications for the Existing System	80
4.2 Operating System Configuration of the Current System	81
4.3 Candidate Matrix for the Proposed System	87
4.4 Projected Costs for the Candidate Systems	88
4.5 Feasibility Matrix for the Candidate Systems	90
A.1 URL Addresses for Software Used in the Project	117
A.2 URL Addresses for Related Organizations	117



# I. INTRODUCTION

## 1.1 Background of the Project

The Internet Protocol version 4 (IPv4) became a Standard in 1981 after years of development dating back to 1969. It has proven to be a very successful protocol, robust and quite scalable, and it is, of course, the basis of the Internet as we know it.

By the early 1990s, however, the explosion in the size of the Internet began to worry network engineers because the enormity of what had come to pass far surpassed anything envisioned by the original protocol designers. Specifically, there was a fear of the exhaustion of the Internet address space, although an equally pressing concern was the overload on routing tables due to the relatively inefficient hierarchy of the class-based addressing system. While trying to solve these specific problems, a number of other less critical issues were also addressed as well, in an effort to take advantage of the intervening years of experience with internetworking.

After a long and contentious period of discussion, a consensus was reached and the Internet Protocol version 6 (IPv6) became a standard in December of 1995, although minor adjustments to the protocol continue to the present. During the last eight years, great progress has been made towards the implementation of IPv6 and related protocols on numerous platforms, and it seems certain that a large-scale transition to the new standard is reasonably imminent.

This transition is certain to be a topic of widespread interest and not a little controversy over the coming years. This project aims to explore the justification for the potentially disruptive move to a new technology, the strategies for implementation of this transition and the down-to-earth realities of setting up a system that speaks IPv6 to the world. As a platform for investigating the last of these topics, this project proposes

to develop an experimental system capable of providing HTTP services in the new environment.

## **1.2 Objectives of the Project**

The objectives of this project are twofold: to examine the nature of the next generation Internet Protocol and the rationales for its adoption, and to explore the problems and concerns with its implementation.

This project was conceived as a hybrid (rather than traditional) systems development project. It is hybrid in the sense that the literature review is extended to form a kind of ‘mini-thesis’ that explains the problems with IPv4 and examines how IPv6 was designed to address these problems. In this section of the report, specific attention is paid to changes in ICMPv6 and other affected protocols, measures to provide support for efficient multi-media streaming and mechanisms for network auto-configuration, in addition to the core concerns over the address space and routing problems. This entails a careful comparative analysis of the two protocols.

The initial section concludes with a look at the strategies developed by the Internet Engineering Task Force (IETF) for deployment of the new protocol, which serves as an introduction to the actual system development effort.

The objective of the latter portion of the project is to evaluate several platforms for an experimental deployment of an HTTP server over an IPv6 network. As this is experimental in nature, the evaluation is based more on the feasibility of technical implementation than on the prospects of profitability for an organization, and to this extent, it differs from the traditional systems development project. This goal led to a two-stage effort: the first to serve web pages over a local area network running on the new protocol, and the second to provide HTTP services over the experimental IPv6

backbone (the 6Bone). Therefore, the objectives include the gathering and compiling the following software:

- (1) An appropriate Operating System
- (2) An IP stack for that Operating System
- (3) Various utilities for implementation and testing
- (4) Software for tunneling through PPP and the IPv4 Internet, and
- (5) IPv6 capable HTTP server software.
- (6) An IPv6 enabled web browser

The overriding goal is to assess the hands-on concerns of an actual implementation of a web server in the IPv6 environment and to deliver a working system.

### **1.3 Scope of the Project**

The scope of the first section of the project includes the IPv4 problem set as conceived by the IETF engineers and the IPv6 solution set that they arrived at to serve as the consensus standard as published in the final specification RFC (Request for Comments). (Deering and Hinden 1995) Scope also includes a discussion of transitional strategies for implementation of the new standard. Alternative solution-set proposals are mentioned in passing, but a detailed discussion of the politics and technical considerations of all the alternative solutions is explicitly beyond the scope of this project.

In the latter section of the project, scope includes a feasibility analysis, which explores the level of maturity and the degree of availability of the following platforms:

- (1) Free BSD using the KAME project IP stack
- (2) GNU/Linux using the 2.4.20 kernel stack
- (3) GNU/Linux using the USAGI project IP stack

(4) Microsoft Windows 2003 Server using the available development IP stack

A cost analysis is also conducted to aid in the construction of a candidate matrix, but a full business cost/benefit analysis is excluded from scope due to the non-commercial nature of the project. Candidate selection is discussed, though other system components such as server software and performance measurement tools are specific to the system selected, and so are not analyzed formally.

The scope of this project includes the gathering, compilation and integration of the selected system components, and a demonstration that the system is working using the Ping6, Traceroute6 and Tcpdump utilities. Finally, project success is defined by serving web pages over the IPv6 network. However, sophisticated performance measurement of the network and the HTTP server are beyond the scope of this project. Likewise, security issues will be discussed *en passant*, but this project is not about security and no attempt will be made to deploy security in the implementation.

#### 1.4 Deliverables

The first part of this project delivers a detailed analysis of IPv4 problems, IPv6 solutions to those problems and the deployment strategy of the IETF. This analysis includes discussions of:

- (1) Addressing
- (2) Headers
- (3) ICMPv6 and Other Protocols
- (4) Multicasting and Flows
- (5) Routing
- (6) Autoconfiguration
- (7) Strategies for the transition from IPv4 to IPv6

The second portion of the project will provide the following deliverables:



- (1) A discussion of platform and software selection including a feasibility analysis, a cost analysis, a candidate system matrix and a discussion of the selection.
- (2) A description of gathering, compiling, installing and implementing the required software.
- (3) Reports on testing the system, both across a LAN and over the 6Bone network.
- (4) A Report on testing the web server through an independent host
- (5) Conclusions and recommendations.



## II. IPv4 PROBLEMS AND IPv6 SOLUTIONS

### 2.1 Overview

IPv4 was (and still is) an excellent protocol, which succeeded beyond anyone's expectations and proved itself simple and scalable. In fact, it was so successful that Christian Huitema notes that it was a victim of its own success (Huitema 1996). The worldwide Internet became such an explosive phenomenon that it introduced problems of scale, especially in the address space and routing tables. These were the most pressing issues that gave rise to the next generation Internet Protocol, but Huitema points out that "years of experience [with IPv4] brought lessons... IPv6 is built on this additional knowledge. It is not a simple derivative of IPv4, but a definitive improvement" (Huitema 1996). The overriding goal was to create a new Internet protocol that is simpler to program and more efficient.

According to Feit, version 6 of the Internet protocol has changed in the following major areas:

- (1) It introduces 128-bit addressing, which may be hierarchically structured to make address allocation and routing simpler and more efficient.
- (2) It simplifies the IP header while providing for optional extension headers, which makes for flexibility and extensibility as new networking functions are added in the future.
- (3) It supports authentication, confidentiality and data integrity at the IP level.

It introduces the concept of 'flows' which can be used to support multimedia and real-time transmission requirements.

- (4) It makes it easy to encapsulate other protocols and provides a mechanism for congestion control when carrying these foreign protocols.

- (5) It provides a new method for automatic address self-configuration and a built-in test for address uniqueness.
- (6) It improves router discovery and the detection of router failure or unreachable neighbors on a link. (Feit 1997)

There have also been numerous minor adjustments of interest, such as the redefinition of the Time-to-Live (TTL) field as a straightforward hop count and the elimination of hop-by-hop fragmentation and the header checksum. Discussion of these streamlined mechanisms will be found in section 2.3.2, which discusses the IPv6 header. Security mechanisms are discussed in 2.3.3 since they are implemented using extension headers.

In passing, one may wonder why this new protocol is numbered as version 6 rather than version 5. The version 5 designation was reserved for the Internet Stream Protocol (STP), which can be examined in RFC 1819, although it never actually came into widespread use.

In this section, we examine the problems that arose with IPv4 as the Internet expanded exponentially and the solutions that the IETF ultimately decided on in response to those problems. The rationales for these decisions are discussed, as are their implications.

## **2.2 The Address Space**

### **2.2.1 IPv4 Addresses**

IPv4 uses 32 bits to encode its addressing scheme. In theory, that number of bits is sufficient to encode some four billion unique Internet addresses, but by the early 1990s, those addresses began to seem scarce even though nothing like four billion computers exist in the world today.

This discrepancy came about because of a flaw in the design of the IPv4 addressing scheme, specifically class-based addressing. To briefly review the IPv4 addressing system, recall that IP addresses are composed of two parts, the network address and the host address. IP classes are defined by the number of bits allotted to the network address as divided on 8-bit boundaries. Thus, we have the following classes:

- (1) Class A addresses, where the network address is eight bits, yielding 128 networks, each with about 16 million subnet addresses.
- (2) Class B addresses, where the network address is 16 bits, yielding some 16,000 potential networks, each of which can contain about the same number of subnet addresses.
- (3) Class C addresses, where the network address is 24 bits, yielding around 2,000,000 available networks, but each of these networks is limited to 128 subnet addresses.
- (4) The remaining classes take their distinguishing bits from the last eight bits available and are used for multicasting (Class D) or reserved for research purposes (Class E).

While this seemed like an ocean of addresses 20 years ago, the potential problem is obvious now.

There are simply too many addresses locked up in class A and class B, which became essentially unavailable a decade ago. Moreover, class C networks don't offer enough interface addresses for many organizational network needs. Organizations that got their class A addresses early are sitting on a wealth of addresses that they could never possibly use, and in all likelihood, most organizations that have class B addresses would be hard pressed to put them all to work. With the possible exception of large ISPs, which distribute their address pool via the Dynamic Host Configuration Protocol (DHCP), those unused addresses are lost forever.

## 2.2.2 Stopgap Measures



In response to this situation, Classless Inter-Domain Routing (CIDR) was introduced, which allows super-netting and sub-netting on boundaries other than 8 bits, permitting much greater flexibility in network address assignment. Elements of the CIDR solution, such as the slash notation for subnet masks, are employed in IPv6.

Another interesting stopgap solution is Network Address Translation (NAT). With NAT, an internal network may be allocated any private addresses that the administrator desires (although for safety reasons it is desirable to use reserved addresses which are not routable, and so can't 'leak' out to the Internet). All outside communication is handled through a gateway server, which has a valid global network address on its external interface. The server keeps track of the local clients' external communication traffic by assigning high port numbers for external TCP connections. Needless to say, the disadvantage here is the amount of overhead on the server and the lack of flexibility, since a server device cannot sit behind a NAT server.

The Dynamic Host Configuration Protocol (DHCP) mentioned above is yet another stopgap measure that allows more efficient utilization of a limited pool of available global IP addresses by allowing addresses to be assigned by a server at boot time or as necessary. DHCP will continue in the IPv6 autoconfiguration mechanism.

For all the ingenuity of these methods, the exponential nature of the Internet growth curve warned of its pending strangulation from a lack of addresses. Huitema, who was deeply involved with the IETF discussions at that time, made an effort to quantify the time span until 'doomsday' when no further Internet growth would be possible. By plotting the estimated number of addresses allocated every month on a log scale, he found that the curve crosses the theoretical maximum of four billion somewhere between 2005 and 2015. (Huitema 1996) However, it is notable that on the one hand, this quantification does not yet take into account the relative inefficiency of

address allocation procedures discussed above, nor does it take into account the ameliorative effects of remedies such as CIDR and NAT on the other. A more nuanced quantification was developed in an effort to determine a desirable address size, and this will be discussed in the next section.

### 2.2.3 IPv6 Address Size

The IPv6 Protocol increases the address space to 128 bits or 16 bytes (octets). Addresses with 128 bits will theoretically allow for  $2^{128}$  IP addresses. The decimal equivalent is 340, 282, 366, 920, 938, 463, 463, 374, 607, 431, 768, 211, 456, a number so large that there is no word for it in any human language. To put this in perspective, this adds up to  $6.65 \times 10^{23}$  network addresses per square meter of the earth's surface, which is of the same order of magnitude as Avogadro's number ( $6.02 \times 10^{23}$ ). To make a contribution to the ever-growing body of anecdotes about how huge this number is, a back-of-the-envelope calculation indicates that this will allow allocation of an interface address for every molecule of a film of water almost 0.19 millimeters deep covering the entire earth. That should hold us for a while.

Such a huge address space may seem like overkill. In an illuminating discussion of the controversy that this choice has engendered, Huitema notes that a considerable number of network engineers who were involved thought that 64 bits would be sufficient. (Huitema 1996) The arguments for the enormously larger address space were: first that the allocation of network addresses is far from perfectly efficient, and also that a vast number of embedded systems will be in need of network addresses in the not-too-distant future. When every mobile phone, kitchen appliance, household control device and automobile part in the world requires its own set of interface addresses, the demand for addresses will very likely be of this order of magnitude.

In order to address the issue of efficiency quantitatively, Huitema formulated a logarithmic efficiency ratio called the H-Ratio, which takes into account the hierarchical dependencies of network addresses. (Huitema 1994)

$$H = \frac{\log(\text{number of addresses})}{\text{available bits}}$$

Because people tend to use powers of 10 when discussing large numbers, the value of H is expressed in base 10 logarithms, and thus ranges in value from zero to a theoretical maximum of 0.30103 (the value of  $\log_{10}(2)$ ). RFC 1715 (Huitema 1994) examines a number of existing example networks and asserts that reasonable values lie between 0.14 and 0.26. Thus we can estimate the population of network addresses for a given address size as given in Table 2.1.

Table 2.1. Estimated Network Address Populations per Address Size.

Address Space	Pessimistic (H = 0.14)	Optimistic (H = 0.26))
32 bits	$3 \times 10^4$	$2 \times 10^8$
64 bits	$9 \times 10^8$	$4 \times 10^{16}$
80 bits	$1.6 \times 10^{11}$	$2.6 \times 10^{27}$
128 bits	$8 \times 10^{17}$	$2 \times 10^{33}$

## 2.2.4 Typographical Expression of IPv6 Addresses

Huitema notes that the “hyperbolic estimate” of the number of Internet hosts over the next 30 years is  $1.0 \times 10^{15}$ , so an address size of 128 bits will serve even if the most pessimistic efficiency is achieved, assuring us that the address space problem will not recur in the remotely conceivable future. (Huitema 1996) As we shall discuss later, other alterations in the IPv6 header have resulted in efficiencies such that the header

size is not greatly increased despite the swollen address fields. Consider that the minimum Maximum Transfer Unit (MTU) in IPv4 is 576 octets and the IPv6 MTU is 1280 octets, while the header length in IPv4 is 20 octets (as a minimum, it can increase to 60 octets with IPv4 options) and in IPv6 it is 48 octets (fixed). This represents 3.4 % of MTU in IPv4 and 3.8 % of MTU in IPv6. Thus, the header overhead is almost equal.

Oddly enough, there were experts who argued for even larger address fields (up to 255 octets) and that these field lengths should be variable. Their reasoning is that the Internet becomes larger and more complex, additional layers of hierarchy will be required, using up even more addressing bits. Therefore, 128-bit addressing represents a true compromise in terms of length. As to variable length addressing, Huitema notes that the allocation and management of memory for variable length addresses would result in software that is larger, slower and more error prone. Furthermore, he adds, “The little experience we have had with the OSI suite and the NSAP [Network Service Access Point] addresses showed us that managers tend to use a fixed-length format even if the protocol allowed variable length” (Huitema 1996).

Initially, only about 15% of the available address space has been allocated for assignment, leaving the remaining 85% for future growth. The structure of the address space will be discussed in a later section.

While IPv4 addresses are typographically represented with the familiar ‘dotted quad’ notation, these very long numbers are expressed as eight groups of four hexadecimal digits each separated by colons. Each group is thus composed of two bytes. This results in addresses of the form:

FDEC:BA98:7654:3210:0BDF:000F:2922:FFFF.



Naturally, this is quite a mouthful, so abbreviations can be used if any of the groups have leading zeros. Trailing zeros do not qualify. The address above thus becomes FDEC:BA98:7654:3210:BDF:F:2922:FFFF.

Initially, most of the groups of an address will consist entirely of zeros, allowing a further compaction of the expression. For example, FDEC:0:0:0:BBFF:0:FFFF becomes FDEC::BBFF:0:FFF. Loopback addresses are an extreme example of this, taking the form ::1. It is notable that only one contiguous series of zeros can be abbreviated in this fashion without creating a fatal ambiguity.

The familiar CIDR slash notation, is also used in IPv6 to indicate the subnet number, e.g. FDEC::BBFF:0:FFFF/60.

This choice of typographical conventions does much to alleviate the cumbersome nature of working with such large numbers, though there is bound to be a considerable amount of complaining as people become familiar with the new system. To answer such complaints, one need only to point to RFC 1924 (Eltz 1996), which suggests that version 6 addresses be represented as base 85 numbers represented by 85 of the 96 printable ASCII characters. This completely official RFC is dated April 1, 1996, which shows that Internet Engineers do have a sense of humor.

### 2.2.5 Categories of IPv6 Addresses

Like IPv4, IPv6 addresses can be split into network and host parts using subnet masks. IPv4 has shown that it is sometimes desirable for more than one IP address to be assigned to an interface, each for a different purpose (e.g. aliases or multi-cast). To remain extensible in the future, IPv6 goes further in allowing more than one IPv6 address to be assigned to an interface. The RFCs do not define a limit to the number of addresses that an interface may have, although implementations of the IPv6 stack may do so to prevent Denial of Service attacks.

Given this large number of bits for addresses, IPv6 defines address types based on sets of leading bits, an approach that will hopefully avoid the situation with IPv4 classes A, B, C and D.

The new protocol defines three main types of addresses: unicast, anycast and multicast. The concepts of unicast and multicast are familiar from IPv4 and other protocols, but anycast is a new feature in IPv6. Basically, it defines a group of computers whose addresses have the same prefix. A packet sent to an anycast address must be delivered to exactly one (and only one) member of the group. This member will be the one that is closest or most easily accessible to the sender as determined by the routing table distance information.

Broadcast addresses are noticeably absent from IPv6, having been subsumed into the multicast address architecture.

## 2.2.6 Address Space Assignment

IPv6 addresses are divided into two segments, with the first segment being a variable-length type prefix. This type prefix contains a code that is designed to assure that no code is identical to the first part of any other code. Table 2.2 on the next page lists the type prefixes according to RFC 2373 (Hinden and Deering 1998).

It must be stressed that this table is the correct one. During the course of this research, it was discovered that the tables given by such authorities as Forouzan (Forouzan 2003), Feit (Feit 1997) and Black (Black 1998) are in error. Calculated comparisons with IPv6 addresses that have actually been assigned demonstrate that engineers are following the RFCs rather than the texts and reference books. The mistaken tables seem to have been borrowed from Huitema (Huitema 1996), who may be forgiven as he wrote before the final standard had been published.

The most recent RFC on this topic, RFC 3513 (Hinden and Deering 2003), omits the set of addresses for IPX protocol networks, lending support to Gai's prediction that "IPv6 will be the only layer 3 protocol of the new millennium" (Gai 1998). However, since one of the innovative design objectives for the new protocol was to carry traffic from any of the network layer protocols, the address allocations for IPX and the OSI Network Service Access Point (NSAP) addresses are included in this table.

Table 2.2. Type Prefixes for IPv6 Addresses.

Allocation	Prefix (binary)	Fraction of Address Space
Reserved	0000 0000	1/256
Unassigned	0000 0001	1/256
Reserved for NSAP	0000 001	1/128
Reserved for IPX	0000 010	1/128
Unassigned	0000 011	1/128
Unassigned	0000 1	1/32
Unassigned	0001	1/16
Aggregatable Global Unicast Addresses	001	1/8
Unassigned	010	1/8
Unassigned	011	1/8
Unassigned	100	1/8
Unassigned	101	1/8
Unassigned	110	1/8
Unassigned	1110	1/16
Unassigned	1111 0	1/32
Unassigned	1111 10	1/64
Unassigned	1111 110	1/128
Unassigned	1111 1110 0	1/512
Link-Local Unicast	1111 1110 10	1/1024
Site-Local Unicast	1111 1110 11	1/1024
Multicast Addresses	1111 1111	1/256

### 2.2.7 Special Addresses

There exists a set of un-prefixed addresses (i.e. all zeros in the prefix space), which serve purposes such as the loopback or local host address ( ::1 ) that is used by a

node to send an IPv6 packet to itself. It may never be assigned to any physical interface and is treated as having link-local scope. It may be thought of as the link-local unicast address of a virtual interface.

The unspecified host identification ( `::` , the all zero address) is mostly seen in socket binding (to any IPv6 address) or in routing tables. It is used at initialization time as the source address for address discovery, which relies on ICMPv6 rather than the old ARP protocol. Note that this unspecified host address must never be used as a destination address.

The other use for un-prefixed addresses is in embedding IPv4 addresses in order to smooth the deployment of the new protocol by allowing it to exist side-by-side with legacy networks. There are two kinds of imbedded IPv4 address prefixes. The first is the IPv4-mapped IPv6 address, which has a special 96 bit prefix length of the form `0:0:0:0:0:FFFF:a.b.c.d/96`, where a.b.c.d represents the IPv4 address. This is used for carrying traffic from systems that do not support IPv6 through IPv6 networks. The reader may refer to section 3.3 for a discussion of tunneling.

The other kind of imbedded IPv4 address is the IPv4-compatible IPv6 address, which takes the form `0:0:0:0:0:0:a.b.c.d/96`. This was designed for tunneling through IPv4 networks (Gai 1998), but is being replaced by a newer mechanism called 6to4 tunneling, which is hierarchically prefixed and will be discussed below in section 3.3.

Unlike these special case addresses, unicast, multicast and anycast addresses all have hierarchically organized prefixes.

#### 2.2.8 Unicast Addresses

Unicast addresses include several categories distinguished according to their scope: global aggregatable addresses, site-local addresses and link-local addresses. Earlier RFCs had reserved a section of the address space for a geographical hierarchy,

but this is absent from more recent documents. One surmises that provider-based hierarchies seem more practical at this transitional stage.

The type of address that is of the most immediate interest to many is the aggregatable global unicast address type, which may also be thought of as provider-based addresses. After the first three-bit type identifier code, five bits are allocated to a registry identifier code. At present, three registry centers have been defined. INTERNIC, the center for North America, has been assigned code 11000, while RIPNIC, the European registry, has code 01000 and APIC, our Asia and Pacific registry has code 10100. Thus the first bytes of provider-based addresses are, in our hexadecimal representation, 0x38, 0x28, and 0x34 respectively. Addresses assigned outside of this registry framework fall in the 2001::/16 range

The remaining 120 bits of a provider-based address contains a provider identifier of 16 bits, a subscriber identifier of 24 bits, a subnet identifier of 32 bits, and a node identifier of 48 bits. Internet service providers are expected to obtain the provider ID from the registries and in turn assign subscriber IDs to their customers. The length of the node identifier was chosen to coincide with the length of an Ethernet link address and will likely be used in the future to contain the node physical address.

These identifiers provide us with an address hierarchy as follows:

- (1) A provider prefix consisting of the type, registry, and provider identifiers.
- (2) A subscriber prefix consisting of the provider prefix and the subscriber identifier.
- (3) A subnet prefix consisting of the subscriber prefix and the subnet identifier.

In this hierarchy, each prefix defines a level of the hierarchy uniquely from type to registry to provider to subscriber to subnet, with the node identifier uniquely defining the host on the subnet.



3 bits	5 bits	16 bits	24 bits	32 bits	48 bits
001	Registry ID	Provider ID	Subscriber ID	Subnet ID	Node ID

Figure 2.1. IPv6 Hierarchical Global Address Structure.

There are two types of local-use unicast addresses defined. These are Link-Local and Site-Local. The Link-Local is for use on a single link and the Site-Local is for use in a single site.

Link-Local addresses are designed to be used for addressing on a single link for purposes such as auto-address configuration, neighbor discovery, or when no routers are present. Routers must not forward any packets with link-local source or destination addresses to other links. The structure of a link-local address is shown in Figure 2.2.

10 bits	54 bits	64 bits
1111111010	00000000000000000000...0	Interface ID

Figure 2.2. IPv6 Link-local Address Structure.

Site-Local addresses are designed for addressing within a site where a global prefix is not needed. An example of a situation where this would prove useful would be an organizational intranet. The format of a site-local address is shown in Figure 2.3.

10 bits	38 bits	16 bits	64 bits
1111111011	0000 00000000...0	Subnet ID	Interface ID

Figure 2.3. IPv6 Site-local Address Structure.

Once again, routers must not forward any packets with site-local source or destination addresses outside of the site. In IPv4, site-local scope is implemented by limiting the Time-to-Live hop count.

### 2.2.9 Anycast Addresses

Anycast addresses are one of the more interesting innovations in IPv6. Examples of the uses of anycast addresses might be 'fuzzy routing' where all the routers at a particular service provider, all the routers at the boundary of an autonomous system (AS) or all the routers attached to a given LAN might share one anycast address.

An IPv6 anycast address is a unicast address that is assigned to more than one interface (typically belonging to different hosts or routers), such that a packet sent to an anycast address is routed to the nearest interface having that address, according to the routing protocol measure of distance. Anycast addresses are created from the unicast address space, using any of the unicast address formats. Anycast addresses are therefore identical to unicast addresses in terms of their syntax. When a unicast address is assigned to more than one interface, it becomes an anycast address. The nodes to which the address is assigned must be configured to know that the address is an anycast address.

Initially, only routers will be assigned anycast addresses, in addition to their normal unicast address, of course. An anycast address may be included in a source

route, indicating that the destination node should use any router that has the same anycast address. This allows for a flexible efficiency in locating the shortest path back to the source.

#### 2.9.10 Multicast Addresses

An IPv6 multicast address is an identifier for a group of nodes. A node may belong to any number of multicast groups. Multicast addresses have the format shown in Figure 2.4.

8 bits	4 bits	4 bits	112 bits
11111111	Flag	Scope	Group ID

Figure 2.4. IPv6 Multicast Address Structure.

The hexadecimal number 0xFF (all ones) in the byte at the beginning of the prefix marks the address as a multicast address.

The first three flag bits are reserved. If the fourth is a one, it indicates a permanently assigned ('well-known') multicast address, assigned by the IANA. If it is a zero, it indicates a non-permanently-assigned ('transient') multicast address.

Scope is a 4-bit multicast scope value used to limit the scope of the multicast group. The values are shown in Table 2.3.

Table 2.3. Interpretation of Scope Values in Multicast Addresses

Value	Interpretation
0	reserved
1	node-local scope
2	link-local scope
3	(unassigned)
4	(unassigned)
5	site-local scope
6	(unassigned)
7	(unassigned)
8	organization-local scope
9	(unassigned)
A	(unassigned)
B	(unassigned)
C	(unassigned)
D	(unassigned)
E	global scope
F	reserved

As mentioned above, IPv6 does not rely on broadcasts for control functions such as address resolution or booting, but rather migrates these to the multicast system. The reasons for this have to do with the inefficiency of having every node on a network process every broadcast packet. All broadcast packets must be opened and their TCP headers examined, which represents a great deal of overhead. With multicasting, only the IP address needs to be examined, and the packet is dropped without further processing if the node is not configured for that multicast address. Another result of this decision is that it will be far easier to provide IP support for non-broadcast multiple access (NBMA) networks such as frame relay or ATM networks.

## 2.3 The IP Header

### 2.3.1 The IPv4 Header

The structure of the IPv4 header is quite familiar, but it will be briefly presented here in order to provide a framework for examining the changes brought about in the IPv6 header. Figure 2.5 provides a graphical guide.

Version	Header Length	Service Type	Total Length	
Identification			Flags	Fragment Offset
Time to Live		Protocol	Header Checksum	
Source Address				
Destination Address				
Options				Padding

Figure 2.5. The IPv4 Header.

The Version field is four bits wide and specifies the version of the IP protocol of the packet. Header length is also four bits and defines the length of the header in terms of the number of 4-byte words. It is needed because the length of the header is variable between 20 and 60 bytes depending on how many Options fields are included.

Service Type, which has recently been changed to Differentiated Services by the IETF, is eight bits wide. The different names imply different interpretations of the bits in the field. With a Service Type field, the first three bits define precedence, which is the priority of the datagram in the event of issues such as congestion. The next four bits define the type of service requested by the datagram. In order of position, a set bit in this field means minimize delay, maximize throughput, maximize reliability or



minimize cost. The final bit is not used. In practice, this field is not much used for reasons that will be discussed in the next section.

Differentiated Services interprets the first six bits of the field as a codepoint whose set of values will be assigned by the IETF, although these have not yet been finalized. This new field is backward compatible with the Service Type field when the first three bits of the field are zeros.

The next four fields may be grouped as Segmentation Control fields. They are used by routers to handle the fragmentation of datagrams in the case that the size of the datagram exceeds the Maximum Transmission Unit (MTU) of the next hop.

Total Length is a 16-bit field that defines the total length of the header plus the payload of the datagram in bytes. The upper limit of this value is 65,535 ( $2^{16} - 1$ ). It is necessary to determine whether fragmentation is necessary in the light of the MTU of a next-hop link.

Identification is a 16-bit field that is also used in fragmentation. This field, along with the source address, must uniquely identify a datagram as it leaves the source host. If the datagram is fragmented in transit, the receiving node can use this to know which fragments belong to which datagram during reassembly.

The flags are three bits that indicate whether a datagram may be fragmented and whether there are more fragments to follow. If the second bit (often called the 'don't fragment' bit) is set, the packet must not be fragmented. Should the MTU of the next link be smaller than the value in the Total Length and this bit is set, the packet will be dropped and an ICMP message will be sent to the source address. The last bit is set to one if there are more fragments to follow. If this is the last or only fragment, it is set to zero.

The Fragmentation Offset is a 13-bit field that indicates the relative position of a fragment in the entire datagram. It represents the offset of the fragment in units of eight bytes, which forces the fragmenting router to choose the size of each fragment so that the first byte number is divisible by eight.

It is notable how much effort is tied up into fragmentation, effort that did not seem entirely justifiable to the IPv6 designers, as will be discussed in the next section.

The 8-bit Time-to-Live field was originally intended to hold a time-stamp, which would be decremented by the amount of time that the datagram spent in each router that it visited. As such a scheme requires synchronized clocks and quite a great deal of processing at each router, almost all implementations treat this field a simple hop count to be decremented by one at each visited router. The purpose of the hop count is to prevent routing loops in the case of routing table corruption. No one wants a packet bouncing back and forth among a set of routers forever or perhaps popping up much later to the confusion of the upper level protocols. It is also useful in limiting the distance that a packet can travel, as when, for example, one wants to confine a packet to a local network. This mechanism is used by the Traceroute utility to discover the path traversed by packets.

The 8-bit protocol field specifies the upper level protocol to which the datagram should be delivered upon receipt by the network layer. A datagram can encapsulate data from a number of higher-level protocols, such as TCP, UDP at the transport layer or ICMP, IGMP and OSPF at the network layer. This field simply tells IP which upper level protocol to hand the datagram off to.

The Checksum field takes up 16 bits and contains a one's complement checksum calculated on the header alone (not the entire packet) at the source. The reason for calculating the checksum in this manner is that the higher level protocols have already

included a checksum on the encapsulated data and that the header changes at every router, so that a checksum on only the header, not the entire datagram, needs to be recomputed at each router. Here again, the IETF designers thought that this effort was hard to justify.

The Source and Destination Address fields are, of course, 32 bits long. Their purpose is self-explanatory. These fields mark the end of the fixed-size portion of the IPv4 header.

Options are used for testing and debugging and are not required in every datagram. They have an internal structure consisting of a 1-byte code field, a 1-byte length field, and a variable-sized data field. The 8-bit code field contains three sub-fields. The Copy field is a 1-bit sub-field that controls the presence of the option in fragmentation. A zero bit means that the option must be copied only to the first fragment, while a set bit means it must be copied to all fragments. The Class field contains two bits and defines the general purpose of the option. The value 00 means that it is used for datagram control, and a value of 10 indicates that it is used for management and debugging. Other values are not defined. The 5-bit Number sub-field defines the type of the option. Of the 32 possible values, only six types are in use.

These option types are:

- (1) No operation - used as a filler between options
- (2) End of option - used as padding at the end of an option
- (3) Record route - used to record up to nine routers that handle the datagram
- (4) Strict source route - used to predetermine the route for the datagram
- (5) Loose source route - similar to strict source route, but other routers may be visited
- (6) Timestamp - used to record the time of datagram processing by a router

Next, an 8-bit length sub-field defines the total length of the option. Finally, there is a variable length data sub-field that contains the data required for a specific option. These last two sub-fields are not present in all option types.

As will be discussed shortly, the IETF decided on an entirely different mechanism to handle the functionality provided by the Options field.

### 2.3.2 The IPv6 Header

Three major simplifications were thought desirable in the design of the IPv6 header: assignment of a fixed format to all headers, removal of the Header Checksum and removal of the hop-by-hop segmentation procedure. This results in the suppression of six fields: the Header Length, the Type of Service, the Identification, the Flags, the Fragment Offset and the Header Checksum. Three fields are renamed and some of these are redefined slightly: the Length, the Protocol Type and the Time-to-Live fields. Two new fields are added: Priority and Flow Label. Finally, the option mechanism is completely revised. Figure 2.6 provides a graphical representation of the IPv6 header structure.

Version	Priority	Flow Label		
Payload Length			Next Header	Hop Limit
Source Address				
Destination Address				

Figure 2.6. The IPv6 Header.

It is immediately noticeable how much simpler the new header is, with only six fields and two addresses compared to the ten fields, two addresses along with the possibility of options in the version 4 header. In this section, the changes and the reasoning underlying them is explored as the header fields are discussed.

The 4-bit version field has a value of 6 (binary 0110) for IPv6, but is otherwise unchanged. The original plan was to run version 4 and version 6 on the same wire and differentiate between them based on this field. It now seems more likely that the two versions will be demultiplexed at the Media Access (MAC) layer, for example by assigning a value of 86DD to the Ethernet content type field rather than the IPv4 value of 8000.

The 4-bit priority field defines the priority of the packet with respect to congestion-controlled traffic, where traffic congestion may increase the amount of time that a packet spends in a router or even cause that packet to be dropped. In the version 4 header, this functionality is handled by three bits in the Service Type (or Differentiated Service) field. The rest of the Service Type functionality has been adapted into the far more comprehensive flow mechanism. Table 2.4 describes the Priorities.

Table 2.4. Priorities for congestion-controlled traffic.

Priority	Interpretation
0	No specific traffic
1	Background data
2	Unattended data traffic
3	Reserved
4	Attended bulk data traffic
5	Reserved
6	Interactive traffic
7	Control traffic



Priorities can also be used for non-congestion controlled traffic when that traffic expects minimum delay. For example, real-time audio and video is not the sort of traffic that can adapt itself to congestion. If packets are dropped, retransmission is not possible. A standard set of assignments for this kind of data has not yet been defined, but a scale of 8 to 15 can be used depending on the amount of redundancy that the data contains.

The Flow Label field is 24 bits long and provides for special handling for a particular flow of data. Flows will be discussed at length in section 2.5.

The IPv4 Header Length field has been eliminated entirely because all headers are of uniform size. The Options field, which is the cause of the variability in IPv4 header length, has been replaced by a system of cascading headers, with extension headers being added to the packet between the main header and the payload.

The Total Length field in the old header has been replaced by a Payload Length field that is 16 bits wide. There is an important difference between the two in that the Total Length field contained the length of the header plus the length of the payload, while the Payload Length field, as its name implies, describes the length of the payload alone. As a result of the size of this field, IPv6 payloads are limited to 64 kilobytes, although provision has been made for 'jumbograms.' Large mainframes and supercomputers on very high bandwidth networks often prefer to work with larger packet sizes so that this provision has been made for them. A detailed description of these jumbo payloads will be found in the discussion of extension headers in section 2.3.3.

IPv6 takes a radically different approach to fragmentation than version 4 did, so the Segmentation Control fields have been moved from the base header to an extension header. Hop-by-hop fragmentation has been eliminated, with only end-to-end

fragmentation allowed. IPv6 routers always operate as if the 'don't fragment' bit were set. The new protocol relies on MTU discovery, where the source host determines the smallest MTU on the path to the destination before sending the packet or flow of packets. If a source chooses not to determine or remember the path MTU, a payload size of 536 octets may be chosen, as IPv6 requires a MTU of at least 576 octets of all links. If a link cannot meet this constraint, it must provide a link layer fragmentation mechanism. Details of path MTU discovery are available in RFC 119.

The Next Header field is eight bits wide and defines the header that follows the base header in the packet. This field is somewhat analogous to the Protocol field of the IPv4 header. It is part of the mechanism to replace the IPv4 Options field with what Huitema calls "a daisy chain of headers" (Huitema 1996). The next header is either one of the optional extension headers used by IP or the header for an upper level protocol such as TCP or UDP. Each extension header also contains this field, as will be seen in the discussion of extension headers below. A list of the next header codes used in this field and the next header type that they specify is given in Table 2.4. Note that the null code (value = 59) specifies the termination of the chain of headers.

Table 2.5. Next Header Codes.

Code	Next Header
0	Hop-by hop option
2	ICMP
6	TCP
17	UDP
43	Source Routing
44	Fragmentation
50	Encrypted security payload
51	Authentication
59	Null (No next header)
60	Destination option

Following the Next Header field is the 8-bit Hop Limit field. This field serves the same purpose as the Time-to-Live field in the version 4 header. As was discussed earlier, the original concept was to use a time value in this field and decrement it according to the amount of time that the packet spent in routers or on the wire. It is not surprising that this proved unfeasible due to the need for synchronous clocking and processing at the router. As a result, the TTL field was more often implemented as a hop count that was decremented by one at each router that it visited. Huitema refers to the name change for this field as “truth in advertising.” (Huitema 1996)

The purpose of a hop count is to prevent routing loops in the case of routing table corruption. It is also useful in limiting the distance that a packet can travel, as when, for example, one wants to confine a packet to a local network. With eight bits, the maximum number of hops is 255, which seems very large. The somewhat dated Routing Information Protocol (RIP) defines infinity as 15 hops, although some routes in the modern Internet can require up to 32 hops. A hop count larger than eight bits was thought to defeat the purpose of the hop count mechanism, but the use of a set hop limit of 255 provides a mechanism to prevent the spoofing of router messages by crackers.

The Source and Destination Address fields, both 128 bits long, are next. This IPv6 header is followed by any extension headers that may be present, and then by the payload, which is, of course, a data segment from an upper level protocol.

Note that the Header Checksum field has been entirely eliminated from the version 6 header in what Huitema describes as “a rather bold move” (Huitema 1996). Other error detection mechanisms exist in the upper layers, notably in TCP, and in lower layers, such as Ethernet or PPP, so the purpose of the header checksum is to catch transmission errors that affect the header but might not be caught by media-level

mechanisms. In practice, however, this seems that this doesn't seem to occur on the links, where the media-level checksums are quite effective, but rather within routers due to defective memory boards or programming errors. Because each router must recompute the header checksum when the Time-to-Live field changes, routers disguise any errors they have made, and so the errors are, in fact, never detected.

The risk of corruption of the header can be analyzed as follows:

- (1) Version field changes result in the packet being discarded
- (2) Priority field changes result in the packet being routed with the wrong priority
- (3) Flow Label field changes result in the packet being routed with the wrong priority
- (4) Length field changes to a larger value result in the packet being discarded
- (5) Length field changes to a smaller value would be detected by the TCP end-to-end checksum
- (6) Payload type changes result in correct delivery, but the packet will be rejected by the destination.
- (7) Hop count changes to a larger value are only a problem in the event of a routing loop.
- (8) Hop count changes to a smaller value may cause the packet to be discarded.
- (9) Source address changes would be detected by the end-to-end checksum if the source address was included in the pseudoheader.
- (10) Destination address changes would normally result in an invalid address and the dropping of the packet, but in the event of a mistaken delivery, the end-to-end checksum would detect it. (Huitema 1996)

These errors only entail a serious risk in the event of systematic programming error on an Internet router, which as we have seen is the type of error that routers are guilty of hiding because they recompute the checksum at each hop. The IETF decided

that other management procedures would be more effective. Furthermore, removing the header checksum allows routers to achieve better performance by eliminating the overhead of checksum processing.

### 2.3.3 IPv6 Extension Headers

The IPv4 Options field was seldom used in practice. The reason why this has turned out to be the case is that router packet forwarding code is very highly optimized to achieve the highest possible performance. The most frequently encountered types of packets are given 'fast path' treatment to increase throughput. Packets with options cannot take the fast path, and applications programmers noticed that their programs suffered a performance disadvantage when options were used. For this reason it has become a good programming practice to use only the simplest packets, avoiding options if at all possible.

Nevertheless, the IETF designers felt that there were good reasons to include options for special treatment of packets in IPv6, so they revised the mechanism in such a way that processing of such options is compulsory.

As we have seen, the mechanism that replaces the Options field in the IPv6 header is a system of chained extension headers. One major advantage of this is that the size of the header is fixed at 20 octets, which saves router processing cycles. The extension headers provide several other examples of the general IPv6 philosophy of avoiding unnecessary processing by intermediate routers.

Fragmentation control and source routing have been moved to the extension headers, and provisions for integrated IP Security are made here.

Most extension headers begin with a Next Header field, which encodes the type of the next header, followed by an 8-bit Length field that contains the length of the current extension header expressed in units of eight octets (64 bits). In turn, this is followed by



an options field that is further segmented according to the type of option. Table 2.5 (above) shows the next header codes, which include codes for the extension headers as well as codes for upper layer protocol headers such as TCP and UDP. A null value in the Next Header field indicates that the current extension header is the last one. The types of extension headers are: hop-by-hop options, source routing, fragmentation, authentication, encrypted security payload and a destination option. This section examines each of these types of extension headers in the order recommended by the IPv6 specifications; however, the actual order of the options in a packet is determined by the initial sender. Receiving nodes will process the extension headers in the order that they are received.

The hop-by-hop option is used when the source wants to send information to every router visited by the packet and may contain control, management or debugging information. The Option sub-field consists of eight bits of Code, eight bits of Length and a variable length block of Data. The Code sub-field is further segmented into two bits of Action, one bit that flags whether the option may be changed in transit, and five bits to indicate the type of hop-by-hop option. At present, only three types of options have been defined: PAD1, PADN and the jumbo payload, although this kind of extension header obviously has the potential to serve as a very flexible tool.

PAD1 is a 1-byte alignment option, which is analogous to the no operation padding in the IPv4 Options field. PADN is similar, but allows a variable number of zero padding bytes.

The only interesting current hop-by-hop option is the Jumbo Payload or 'Jumbogram' option that was discussed previously. If a payload longer than 65,535 bytes is required, the Jumbo Payload option defines the length of the payload. The Length sub-field has a fixed value of 4 describing the length of the Data sub-field, so

that the maximum size of a Jumbogram is  $2^{32} - 1$  or something over 4 billion bytes. This indeed would be a pachyderm-sized packet.

The Source Routing extension header combines the strict source route and the loose source route option types of IPv4, with a Type field following the Next Header and Header Length fields to differentiate between the two. The Addresses Left field contains the number of hops remaining to reach the destination, and the Strict/Loose Mask field determines the rigidity of the routing: Strict indicates that the specified route must be followed exactly and Loose means that other routers may be visited in addition to those specified. The structure of a Source Routing extension header is shown graphically in Figure 2.7.

Next Header	HeaderLength	Type	Addresses Left
Reserved	Strict/Loose Mask		
First Address			
Second Address			
.			
.			
.			
Last Address			

Figure 2.7. Format of a Source Routing Extension Header.

The destination address in source routing is not the final destination of the packet, but rather changes from router to router. The router at a given hop opens the Source Routing extension header and uses the Addresses Left field as an index into the list of addresses. The address thus found is placed in the destination address of the base header, and the previous destination address is inserted into that same location in the list

of addresses in the extension header. Thus, both the base header and the extension header are changed at every hop.

The Fragmentation extension header contains the Fragmentation Offset and Fragmentation Identification fields that are in the IPv4 header. These fields are needed in order to allow reassembly of fragments at the destination node. Fragmentation in IPv6 is identical to that in version 4 with the exception of where the fragmentation takes place. As discussed previously, IPv6 does not allow hop-by-hop fragmentation. End-to-end fragmentation using MTU discovery is the rule, and packets that are too large for some hop will simply be dropped and an ICMP message returned to the source address.

It is important to understand that an IPv6 packet consists of a fragmentable portion and one that cannot be fragmented. The unfragmentable part consists of the base header and all the extension headers that must be processed by each node on the path. In practice, these are all the headers up to and including the Source Routing extension header. The fragmentable part is the rest of the packet, i.e. the extension headers that need to be processed only at the destination node and the payload. Thus, all the fragmented packets resulting from a fragmentation at the source host will consist of the unfragmentable series of headers followed by a Fragmentation extension header, which in turn is followed by a fragment of the fragmentable portion of the original packet.

The format of the rest of the Fragmentation header after the Next Header and Header Length fields is comprised of a 13-bit Fragment Offset field, an 8-bit reserved field, a 1-bit More Fragments field and a 32-bit Fragment Identification field. The fragment offset is expressed as the number of 8-octet (64-bit) units by which the current fragments contents are offset from the beginning of the fragmentable part of the original packet before it is fragmented. The More Fragments bit is set to zero for the last

fragment and to one if there are more fragments to follow. The Fragmentation Identification field is identical to that in IPv4.

The Authentication and the Encrypted Security Payload (ESP) extension headers are a result of the effort to provide support for Internet Security (IPSec) at the IP level. IPSec in IPv4 was entirely optional, developed long after the Internet came into existence and integrated into IP in a haphazard, *ad hoc* fashion. In IPv6, IPSec is tightly integrated into the design of the protocol. All implementations must support it, which removes any possible excuse for failing to implement good security policies.

The two functions of the Authentication extension header are to validate the message source and to ensure the integrity of the data. The former is to ensure that the message is from the genuine sender, while the latter is to ensure that no one has altered the data in transit. This header is quite simple, consisting of only two fields. The Security Parameter Index identifies the algorithm used for authentication since a number of algorithms are available, while the Authentication Data field contains the actual data that the algorithm has generated. When calculating the authentication data, all fields that will change during routing, such as the hop count, and the Authentication Data field itself are set to all zeros. The entire resulting IP packet is fed into the algorithm along with a 128-bit security key and the output is written into the Authentication Data field.

At the receiving node the process is reversed. The Authentication Data field is copied and set to all zeros, as are the changeable header fields. The resulting packet is run through the algorithm with the 128-bit secret key and the result is compared to the original Authentication Data field. If they match, the packet is authentic. If they do not match, the packet is dropped.

The purpose of the Encrypted Security Payload (ESP) extension header is to provide confidentiality for the data. The format is similar to that of the Authentication header; with a Security Parameter Index that identifies the type of encryption used and an Encrypted Data field that contains the encrypted data.

There are two modes for transporting encrypted data over IP: the Transport mode and the Tunnel mode. In Transport mode, a TCP segment or UDP user datagram is encrypted and then encapsulated in an IPv6 packet. This packet consists of the base and other headers in plaintext, the Security Parameter Index and the encrypted data as the payload. This is the preferred mode for host-to-host transmission of confidential data.

Tunnel mode encrypts the entire IPv6 packet including its headers and then encapsulates the whole encrypted packet into a new packet with plaintext headers. Such a packet will consist of a plaintext header, a Security Parameter Index and an encrypted packet that includes the original packet's headers in encrypted form. Tunnel mode is mostly used by security gateways to encrypt data.

The last option, the Destination option, is a generic way to add functionality to IPv6 without having to define a new extension header type. There are only 256 header numbers available, so the definition of extension header types must be carefully controlled.

The Destination option is used when the source node needs to pass information to the destination node and doesn't want intermediate routers to be able to access this information. The format of this extension header is identical to that of the Hop-by-hop Options extension header, except that only the PAD1 and PADN option types have been defined.

Following Forouzan, the differences between the IPv4 and IPv6 options can be summarized as follows:



- (1) The No-operation and End-of-option options in IPv4 are replaced by PAD1 and PADN options in IPv6.
- (2) The Timestamp option is not implemented because it is never used.
- (3) The Record Route option is not implemented because it is seldom used.
- (4) The Authentication extension header is new in IPv6.
- (5) The Encrypted Security Payload extension header is new in IPv6.
- (6) The Source Route option is called the Source Route extension header in IPv6.
- (7) The fragmentation fields in the base header section of IPv4 have moved to the Fragmentation extension header in IPv6. (Forouzan 2003)

## **2.4 ICMPv6 and Other Protocols**

### **2.4.1 The Impact of IPv6 on Other Protocols**

Due to the increase in the IPv6 address size, almost every Internet-related protocol and application suite will need to be modified to some degree. With regard to upper layer protocols, the pseudoheader mechanism in TCP must be revamped to reflect the increased IP address size, while the optional UDP checksum mechanism has become mandatory now that the Header Checksum field has been removed from IP.

The most radical change impacts the set of network layer protocols. IPv4 utilizes the Internet Control Message Protocol (ICMP), the Internet Group Management Protocol (IGMP), the Address Resolution Protocol (ARP) and the Reverse Address Resolution Protocol (RARP), all of which are layer 3 protocols in the OSI reference model. In IPv4, RARP has been eliminated altogether as BOOTP has effectively made it obsolete. IGMP and ARP have been consolidated into ICMP, leaving IPv6 and ICMPv6 as the only network layer protocols. The new ICMP is discussed in this section.

### 2.4.2 ICMPv6 Overview

ICMPv6 messages, as in version 4, are divided into two fundamental categories: error-reporting messages and query messages. The general format of ICMP messages starts with a Type field that indicates which of these general categories the message belongs to. This is followed by a Code field, which specifies the purpose of the message. After the Code field, there is a Checksum field, which is calculated in the same way as in version 4. Further information and data come after these fields, as illustrated in Figure 2.8.

Type	Code	Checksum
Other Information		
Rest of the Data		

Figure 2.8. General Format of ICMPv6 Messages.

### 2.4.3 ICMPv6 Error Reporting Messages

One of the primary responsibilities of ICMPv6 is the reporting of errors, which include: destination unreachable, packet too big, time exceeded, parameter problems, and redirection. Source quench has been eliminated in version 6, while the packet too big message is new. Destination unreachable is unchanged from version 4 and uses the format shown in Figure 2.9.

Type: 1	Code: 0 to 4	Checksum
Unused (All zeros)		
Part of the received IP packet, including the IP header and the first 8 bytes of the data		

Figure 2.9. Format of Destination Unreachable Messages.

The codes in the Code field explain the reason for discarding the packet. They are specified as follows:

- (1) Code 0. No path to destination.
- (2) Code 1. Communication is prohibited
- (3) Code 2. Strict Source Routing is not possible
- (4) Code 3. Destination address is unreachable.
- (5) Code 4. Port is not available.

Note that the IP header is returned to the source machine so that it can identify which of its transmissions has caused the error. Likewise, the first eight bytes of the data are included, which would normally be a portion of the upper layer protocol segment header.

The packet too big message is a new addition to ICMPv6. It is a part of the mechanism for the conversion from hop-by-hop to end-to-end fragmentation. If a router receives a packet that is larger than the MTU of the next hop, the router discards the packet and this type of ICMP error packet is sent to the source. There is only one code and the MTU field lets the source node know the maximum size packet acceptable to that network. The packet too big message format is shown in Figure 2.10

Type: 2	Code: 0	Checksum
MTU		
Part of the received IP packet, including the IP header and the first 8 bytes of the data		

Figure 2.10. Format of Packet Too Big Messages.

Time exceeded messages are very similar to those employed in version 4, though the Type value is changed to three. When a router receives a packet with a hop count value of zero, it drops the packet and sends this type of ICMP message to the source address with a Code value of zero. When fragments of a packet are dropped because other fragments have not arrived within the time limit, this type of error message is sent to the source address with a value of one.

Likewise, parameter problem error messages now have a Type value of four, the offset pointer field has been increased to four bytes and the Code field is increased to three possible values instead of two. The Code field identifies the cause of the failure as follows:

- (1) Code 0. An error or ambiguity exists in one of the header fields. With this code, the pointer field points to the byte with the problem.
- (2) Code 1. There is an unrecognizable extension header.
- (3) Code 2. There is an unrecognizable option.

The purpose of the redirection message is to update the source routing table, and the format is similar to that of version 4 with the exception that the format of the packet has been enlarged to accommodate IPv6 addresses. Figure 2.11 illustrates this new format.

Type: 137	Code: 0	Checksum
Reserved		
Target (router) IP address		
Destination IP address		
Option code	Option length	
Target (router) physical address		
Part of the received IP packet, including the IP header and the first 8 bytes of the data		

Figure 2.11. Format of Redirection Messages.

#### 2.4.4 ICMPv6 Query Messages

The query category of messages is also the beneficiary of IETF alterations. Echo request and reply and router solicitation and advertisement have been retained. Timestamp and address mask requests and replies have been suppressed as the former was never used and the IPv6 addressing scheme obviates the need for the latter. New message types have been created to subsume ARP, for neighbor solicitation and advertisement, and IGMP, for group membership management.

With echo request and reply messages, the only change has been in the value of the type field. Echo request and echo reply messages are designed for diagnostic purposes. The combination of the two messages allows network managers to determine whether two systems, whether they are hosts or routers, are able to communicate with each other. This mechanism is the basis of the Ping utility, which is so handy in determining whether a host is reachable and providing delay statistics. Type number 128 indicates a request and 129 indicates a reply. Figure 2.12 shows the echo message format.



Type: 128 or 129	Code: 0	Checksum
Identifier		Sequence Number
Optional data		
Sent by the request message and repeated by the reply message.		

Figure 2.12. Format of Echo Request and Reply Messages.

Traceroute dumps of both IPv4 and IPv6 Ping headers can be found in Appendix B. These dumps allow for a far more detailed and concrete inspection of the packet headers.

Router solicitation and advertisement messages are identical in purpose and similar in format to version 4 messages. An option to allow a host to announce its physical address has been added in order to make it easier for the router to respond. Figure 2.13 gives the format of a router solicitation message.

Type: 133	Code: 0	Checksum
Unused (All zeros)		
Option code: 1	Option length	
Host physical address		

Figure 2.13. Format of Router Solicitation Messages.

The router advertisement format differs from that in ICMPv4 in that the router announces only itself. Several options may be appended to the message packet. One option allows the router to announce its physical address for the convenience of the host. Another is for informing the host of the MTU size of the router. The third option lets the router define its preferred lifetime, which is the number of seconds that an entry in the host routing table for that router should be considered valid.

Figure 2.14 shows the format of router advertisement messages.

Type: 134	Code: 0	Checksum
Hop limit	M O Unused	Router Lifetime
Reachability lifetime		
Reachability transmission interval (timer)		
OPT code: 1	OPT length	
Router physical address		
Option code:5	Option length	Unused (All zeros)
MTU size		

Figure 2.14. Format of Router Advertisement Messages.

The 1-bit M (managed address configuration) field indicates that nodes receiving the advertisement must use the Stateful protocol for autoconfiguration as well as the stateless address autoconfiguration. Similarly, the 1-bit O (other stateful configuration) field tells the receiver that they must use the Stateful Configuration Protocol for additional information. This protocol will be discussed in section 2.7.2, which deals with autoconfiguration.

In IPv6, neighbor discovery messages, which include both neighbor solicitation and advertisement messages, take over the duties of IPv4's Address Resolution Protocol (ARP). With due allowance for the increased address size, the principle is exactly the same. An IPv6 node transmits neighbor solicitation messages to request the link layer addresses of target nodes while also providing the target nodes with its own link layer address. These messages are sent to multicast addresses when a node needs to resolve

an IPv6 address to a link layer address, and to unicast addresses when a node needs to verify the reachability of a neighbor.

The source address of a neighbor solicitation message is either the unicast address of the interface that transmits the message, or the unspecified address when the Dynamic Host Configuration Protocol duplicate address detection procedure is being utilized to ensure that a tentative address is unique. Further discussion of this latter procedure may be found in section 2.7, which discusses autoconfiguration.

The format of a neighbor solicitation message is given in Figure 2.15.

Type: 135	Code: 0	Checksum
Unused (All zeros)		
Target IP address		
Option code: 1	Option length	
Solicitor physical address		

Figure 2.15. Format of Neighbor Solicitation Messages.

A neighbor advertisement message is transmitted upon receipt of a neighbor solicitation message and whenever the state of a node changes in order to propagate knowledge about those changes through the network. The format of the neighbor advertisement is quite similar to the solicitation format except that there are three additional 1-bit fields following the Checksum field. The first of these is the R (router) flag that is set to one if the source is a router. The second is the S (solicited) flag that indicates that the message is being sent in response to a neighbor solicitation message.

The last of these is the O (override) flag which is set to indicate that the message should update the cached link layer address.

The only other differences are that the Type value is 136, the Option code is 2 and the Solicitor physical address is replaced by the target physical address. Neighbor discovery messages can include zero, one or more options. Some options can appear multiple times in the same message. Examples of these include the source/target link layer address option and the prefix information option.

As ICMPv6 is subsuming the role of the Internet Group Management protocol, it has acquired three new group membership message types: report, termination and query. The report and termination messages are sent from the host to the router, while the query message is sent from the router to the host. The first two of these are nearly identical in format, differing only in their Type values, 131 and 132 respectively. The query message replaces the first half of a reserved field following the Checksum with a field for maximum response delay. Its Type value is 130. Figure 2.16 shows the group membership query message format. Multicasting itself will be discussed in some detail in section 2.5.

Type: 130	Code: 0	Checksum
Maximum response delay		Reserved
IPv6 multicast address		

Figure 2.16. Format of Group Membership Query Messages.

## 2.5 Multicasting and Flows in IPv6

### 2.5.1 IPv6 Multicasting

Multicasting was formally added to the IPv4 protocol in 1988 with the addition of Class E addresses. Its implementation was accelerated by the arrival of the Mbone (Multicast Backbone) in 1993, but multicasting deployment is still somewhat sketchy due to the fact that it is an add-on afterthought to the protocol. With IPv6, multicasting becomes an integral part of the core protocol. We have seen the multicast address structure in the Section 2.2.10 and examined the way in which ICMPv6 has subsumed the IGMP in Section 2.4.4.

Recall that multicast addresses begin with a string of eight ones (0xFF), followed by four bits of flags, four bits of scope and a 112 bit group ID. Only the last flag bit contains meaning, and it defines whether an address is a permanent ('well-known' in Internet parlance) address assigned by the IANA or a transient address, which is not permanently assigned. The scope bits define a code (see Table 2.3) that is used to limit the scope of multicast packets so that they do not, for example, leak out into the Internet at large if they are privately scoped. The same functionality was obtained in IPv4 Mbone by carefully tuning the Time-to-Live hop count. However ingenious this may be, it seems rather inelegant. The IPv6 specification allows for much more precise scoping by obliging routers to enforce scope boundaries.

The IPv6 specification defines four permanent group identifiers that all nodes must understand:

- (1) The group identifier 0 is reserved and cannot be used with any scope.
- (2) The group identifier 1 defines all IPv6 node addresses.
- (3) The group identifier 2 defines all IPv6 router addresses.



- (4) The group identifier 0x10000 (hexadecimal) defines the group of all dynamic host configuration servers such as DHCP servers and their relays.

These are used in conjunction with scoping boundaries to identify all nodes, routers or configuration servers on a given node or link. A range of multicast addresses from FF02::1:0:0 to FF02::1:FFFF:FFFF is reserved for ICMPv6 address resolution.

### 2.5.2 Multicast Routing in IPv6

Multicast routing is the routing of packets whose address is a multicast address, i.e. the address of a group of stations. Some multicast addresses are associated with predefined groups or only have a meaning with regard to a node or link. Other multicast groups can have members spread out over the entire Internet. Packets addressed to this latter type of group must be routed.

In IPv4, group membership was administered by IGMP, which has migrated to become an integral part of ICMPv6, and multicast packets were routed using either the Distance Vector Multicast Routing Protocol (DVMRP) or the Multicast Open Shortest Path First (MOSPF) protocol. In order to route multicast packets, a distribution tree must be created to reach all members of the group. Since new members may join a group or existing members may leave, this tree must be dynamic. Addition of members causes the tree to grow, while departure of members causes the tree to be pruned.

In IPv6, the MOSPF extensions were integrated into the OSPFv6 protocol. In order not to drift too far out of the scope of this study, it will be sufficient to say that multicast routing is an integral part of IPv6, in particular of the ICMPv6 and OSPFv6 protocols.

### 2.5.3 Multimedia and Flows

Discussion of multimedia traffic falls naturally into this section on multicasting because multicasting will be the carrier of a large portion of the multimedia traffic

transmitted over the Internet. Gai states that experiments such as the Mbone “have highlighted the intrinsic multicast nature of multimedia traffic” (Gai 1998). Heretofore, the IPv4 Internet was essentially a carrier of best-effort traffic, making no guarantee of packet delivery time or even of delivery at all.

The IPv6 header introduced the Flow Label field as a part of the effort to change this situation. However, the concept of flows is not limited to multicasting. A flow is defined in the IPv6 specification as “a sequence of packets sent from a particular source to a particular (unicast or multicast) destination for which the source desires special handling by the intervening routers” (Huitema 1996). A flow is a set of packets that come from the same source to the same destination and carry the same flow label.

Thus, it is the combination of the provision of flow labels and priorities in the base header, along with the availability of a large reservoir of multicasting addresses that is responsible for the IPv6 promise of improved support for real-time traffic such as streaming multimedia. IPv6 is a part of a larger project called the IS (Integrated Service) Internet which aims to extend the Internet architecture to allow it to carry real-time traffic as well as best-effort traffic, and to control the utilization of transmission links.

Multimedia applications often generate real-time traffic that is sensitive to queuing delays and to packet losses due to congestion. Thus, such traffic needs a guaranteed minimum bandwidth, so IPv6 introduces the concept of Quality of Service (QoS) and the extended service model, which is also referred to as IS (Integrated Services). The traffic control mechanisms of an extended service model router are considerably more complex than those of a traditional router, particularly in the areas of packet scheduling and buffer management. A brief summary of these two routing duties may be considered here.

A packet scheduler determines the order in which each packet is retransmitted. Traditional routers order packets in terms of their priority, which has tended to break down into first-come-first-served or FIFO queuing. The queuing of real-time packets has shifted to WFQ (Weighted Fair Queuing) as the algorithm of choice. Each queue is associated with a weight proportional to the frequency with which it must be served. WFQ alternates the transmission of packets belonging to several flows according to the weights assigned to their queues. This works much like a low-pass filter.

Buffers in a router are necessary whenever packets arrive at a speed higher than they can be retransmitted. If this condition persists, packets will simply have to be dropped. The packets to be discarded should not be selected randomly, but rather as a function of the type of application and of the services that they require. Thus, specific buffer management mechanisms for different types of packets must be implemented. This is not a straightforward process since TCP regards the loss of a packet as a sign of network congestion and will reduce packet generation at the source. Paradoxically, dropping a packet from a full buffer can contribute to the desired QoS because it shortens the delay of the packets that follow the one discarded.

Obviously, this discussion points to the need for a method of packet classification. Since we have the means to identify a flow using the Flow Label header, it is possible to identify the type of application traffic that the flow carries by opening the upper layer protocol header of a flow packet at the time that the flow is initiated. Trying to make decisions based on header content allows QoS service for existing applications without modification. Using the Flow Label and the Priority fields yields an even simpler, if not quite so adequate solution.

#### 2.5.4 RSVP – The Resource Reservation Protocol

The IETF has selected RSVP (the Resource Reservation Protocol) as the layer 3 protocol that allows the network to propagate the resources requested by applications. RSVP is also a pun on the French language expression *repondez s'il vous plait*, as this acronym often appears at the bottom of invitations that require a response.

RSVP supports resource reservations for both unicast and multicast applications and dynamically adapts itself to variations in groups and routing paths. It is used by a host to request a specific QoS from an application and by routers to retransmit QoS requests along the entire routing tree and to maintain state information about the flows. RSVP is for simplex data flows in that the sender is treated differently than the receiver and the RSVP request is unidirectional with the receiver responsible for issuance.

In the RSVP admission control procedure, the receiver must specify the set of necessary resources, called flowspecs, and the set of packets to which the resources are to be allocated, called filterspecs in a request message, which is called a Resv message. This Resv message visits all the routers on the path to the destination, and the resource allocation is negotiated at each router. If the resource is unavailable due to either congestion or policy constraints, the Resv message is dropped. If the resource allocation is agreed upon, the router's state is updated with a definition of the class of the flow and the necessary buffers are allocated. This proceeds stepwise until all intervening routers have entered into the agreement so that the necessary state information is created and managed in a distributed form all along the multicast tree. The sender is limited to messages called Path messages, which inform receivers about the type of transmissions to be made. Thus, RSVP messages only contain control information about flows, not the actual data itself, which is encapsulated in flows of multicast packets.

RSVP currently can use three different reservation styles: wildcard reservation, fixed filter reservation and shared filter reservation. Wildcard reservations create a single reservation shared by all the senders' flows. This is most appropriate for audio flows such as audio conferencing, where a limited number of sources are active at the same time and can share the same resources. In a fixed filter reservation, a dedicated reservation for a particular sender is requested. This cannot be shared, even by other senders in the same multicast group. Video flows typically request this style of reservation. With shared filter reservations, a shared reservation for a set of senders that are explicitly identified is requested. This is also used for audio applications as an alternative to Wildcard style reservations.

It is important to note that not all Internet engineers are enthusiastic about RSVP. The placement of state on routers goes against the basic principle of Internet architecture, and Huitema argues forcefully for adaptive applications that can adjust themselves to changing conditions as a preferable alternative to reservations (Huitema 1996).

## 2.6 Routing in IPv6

Recall that the major problems with IPv6 were the threat of exhaustion of the address space and the collapse of the routing system due to the explosion in size of router tables. These problems, in turn, were largely caused by the inefficient hierarchies that arose from the Class-based allocation of network addresses. The IPv6 addressing scheme should prove itself vastly more efficient, with its registry ID, provider ID, subscriber ID, Subnet ID and Node ID hierarchy (see Figure 2.1).

A geographical hierarchy was also proposed and address space allocated for it, but the free market nature of the Internet seems to militate against a simple geographical solution. Provider-based addressing, however, seems to threaten the possibility of a



tyrannical monopoly, where organizations are locked into providers because of the difficulty of changing the addresses in large networks of machines. However, we shall see that provider-based addressing seems to hold out the best hope for a rational address hierarchy that is needed to keep backbone routing tables down to a manageable size. Autoconfiguration, which will be discussed in section 2.7, lessens the fear of lock-in by Internet service providers.

Static routing refers to the manual configuration of a router's routing table by the administrator and is certainly possible in IPv6. However, static routing is of only limited interest, so only dynamic routing protocols will be considered in this section.

#### 2.6.1 Internal Routing Protocols

The cornerstone of the Internet is the Autonomous System (AS), which is broadly defined as an internetwork under the administrative control of a single organization. The network routing protocol used within an AS is decided by that organization and may be chosen from a set of protocols that is referred to as Internal Routing Protocols. Internal protocols have evolved as networking became more advanced and come in both open and proprietary forms. Two main categories exist for internal protocols, distance vector protocols and link state protocols.

The distance vector algorithm was the first distributed routing algorithm to be implemented. In routers that use the distance vector algorithm, a data structure called the distance vector for each line in the routing table is maintained. This structure contains the address of each destination on the network and the cost metrics associated with it. Each router sends its routing table to neighbor routers in the form of distance vectors. When a router receives a routing table from a neighbor, it compares that table with its own stored routing table. Any new destinations are merged into its routing

table, while already known destinations are examined for changes in state and lower cost metrics in order to keep the tables congruent and up-to-date.

An early example of a distance vector protocol is the Routing Information Protocol (RIP). This venerable protocol is considered by many to be out-of-date, due to its slowness in router table state convergence, its dependence on hop count as the sole cost metric and its limitation of 16 hops. It is, however, simple and reliable for small networks, and a version of it, RIPv6, has been implemented for IPv6 routing.

Notable among the proprietary distance vector protocols are Cisco Systems Interior Gateway Routing Protocol (IGRP) and extended IGRP (EIGRP). These are more capable routing protocols that overcome many of the limitations of RIP by using a system of cost metrics that can include such factors as line speed. It is expected that Cisco will include IPv6 support for EIGRP.

Link state routing protocols are computationally more intensive, but put less overhead on the network than distance vector protocols. Each participating router advertises only its immediately connected links, both periodically and when the state of a link changes, using link state advertisements (LSAs). LSAs are propagated throughout the network by flooding, where routers send out the routing information packets through all of their interfaces. Receiving routers retransmit all information packets that they have received through all interfaces other than the interface where the information packet was received. This information is stored in a database by all receiving routers, and each router computes the topology of the network using the Dijkstra algorithm. Link state routing protocols are inherently more stable than distance vector protocols because each router maintains knowledge of the state of the entire network. Advertising link state changes results in faster time to convergence. However,

link state protocols are more complex and require better equipment to handle the computation involved.

Open Shortest Path First (OSPF) is a hierarchical link state interior routing protocol. The root of the hierarchy is the AS, which can be divided into areas, each one of which contains a group of interconnected networks. OSPF routers can be classified into four non-mutually exclusive categories: internal routers, area border routers, backbone routers and AS boundary routers. Internal routers connect subnets belonging to the same area and use only one instance of the OSPF algorithm. An area border router connects one or more areas to the backbone area, and uses an instance of the OSPF algorithm for each directly connected area as well as one instance for the backbone. These routers collect reachability information from the areas and distribute it to the backbone, which in turn distributes it to other areas. Backbone routers are any routers with an interface on the backbone, so that area border routers are included in the backbone router classification. If all router interfaces are on the backbone, a router is considered to be an internal router. AS boundary routers exchange router information with routers belonging to other ASs.

OSPFv6 is the interior routing protocol recommended for IPv6. It is simply OSPF adapted to 128-bit addresses without any additional functionality added because it is already state-of-the-art. It is layered directly into IPv6, using the value 89 in the Next Header field of the ICMPv6 header.

Another sophisticated internal routing protocol is the integrated Intermediate System to Intermediate System Protocol (IS-IS), which was developed by the International Organization for Standardization (ISO) for its Open System Interconnect (OSI) Connectionless Network Protocol project. IS-IS has been extended to handle IPv4 routing and is expected to be adapted for IPv6 as well.

### 2.6.2 Exterior Routing Protocols

While IPv6 interior routing protocols are, for the most part, simple adaptations of existing protocols to the larger address space, the routing mechanism between autonomous systems has received an overhaul in IPv6. In fact, the very expression autonomous system is being dropped in favor of 'Routing Domain.' The Border Gateway Protocol (BGP) proved to be too heavily optimized for 32-bit addressing, and so has been replaced by Inter-Domain Routing Protocol (IDRP), which was originally designed for use on the OSI architecture for the ISO CLNP protocol, and is, in fact, derived from BGP. It includes all BGP-4 (BGP version 4) functionality and is based on the same path vector philosophy. IDRPv2 is the version designed for operating over IPv6.

Path vector algorithms are similar to distance vector algorithms. In place of cost metrics, however, they advertise a list of Routing Domains to be crossed to reach each destination. Routing Domain lists provide an easy way to check for possible routing loops on the network. In addition, they make policy based routing possible by specifying the Routing Domains to be traversed.

An important difference between BGP and IDRP is that in BGP routing messages are carried over TCP, which introduces interactions between TCP flow control and error detection strategies and BGP routing decisions. In contrast, IDRP uses bare IP datagrams and has its own control mechanisms. Another difference is that IDRP may carry several kinds of addresses, while BGP is strictly an IP protocol.

With autonomous systems, AS numbers were assigned by the IANA. Because Routing Domains are identified by an IPv6 address prefix, AS numbers are no longer

necessary, so the IANA will be relieved of this burden. In addition, this implies that there are as many IDRP Routing Domains as there are addresses. BGP encoded ASs on 16 bits, limiting the number of such systems to 65,536. Routing Domains may be grouped into a Routing Domain confederation, which is viewed as a unique entity. Confederate Routing Domains can also be identified by IPv6 address prefixes, and may contain an arbitrary number of levels of hierarchy.

Routing Domains are divided into two types in IDRP: End Routing Domains (ERDs) and Transit Routing Domains (TRDs). ERDs are Routing Domains in which routes are computed primarily to provide intra-domain routing services, whereas TRD routes are computed primarily to carry inter-domain or transit traffic. This concept is similar to that of stub and transit networks. ERDs are associated with network end user organizations that usually have connections with only one TRD, although sometimes an ERD is connected with multiple TRDs in a multihomed configuration for purposes of redundancy or cost. TRDs are associated with Internet Service Providers (ISPs).

Each IDRP router computes its preferred routing toward a given destination and transmits a path vector information packet to its IDRP-adjacent routers. This computation is subject to the policy that can be configured in each individual router. Furthermore, IDRP allows for multi-protocol routing with multiple address structures. IDRP is layered directly on IPv6 and uses the value 45 in the Next Header field of the ICMPv6 header.

The impact of this exterior routing scheme on the Internet as a whole is of interest. ISPs can be categorized as direct service providers that connect end users to international backbones via themselves or as indirect service providers that connect only indirect service providers and large-scale users. The latter category consists of the highest level of the hierarchy, organizations that directly administer the backbone.



Because direct service providers are allocated a set of addresses that it further divides into smaller sets to be assigned to its users, these sets of addresses can be hierarchically grouped according to the provider's own definition. For ERD routers, the routing table situation remains similar to that in IPv4. Each router has a routing table entry for each network within the ERD and a default entry toward the TRD. ERD routers announce their set of addresses to the TRD with only one entry.

Indirect service providers, however, have a completely changed situation since each direct service provider announces all its networks with only one entry. The routing table size has become proportional to the number of service providers rather than the number of networks as in IPv4. This represents an enormous breakthrough in the problem of backbone router overload.

As mentioned above, other aggregation schemes have been considered, notably geographically based aggregations. The rapidly developing free market in telecommunications makes such schemes seem of questionable feasibility, especially in the light of the advantages of a provider based hierarchy.

However, this provider based hierarchy offer some potential disadvantages for end users. Organizations have come to think that they own their IP network address. With a provider based addressing hierarchy, this is emphatically not the case. The provider owns the address, and any organization that decides to switch providers must be prepared to reconfigure all the hosts and routers on their networks. How IPv6 proposes to address this situation is the topic of the next section.

## **2.7 Autoconfiguration**

Automatic configuration, or autoconfiguration, refers to the ability of computers to discover and register the information that they need in order to be connected to and communicate over a network. Because it is possible that addresses will be assigned to

an interface for a limited lifetime and an interface may have multiple addresses in IPv6, dynamic multiple address configuration is an integral part of the IPv6 design standard.

There are two modes of autoconfiguration: stateless mode, which configures a machine to communicate with other machines over the same link, and stateful mode, which uses DHCP servers to assign addresses to a large network.

### 2.7.1 Link Local Addresses

Recall that link local addresses have a 64-bit field for the device ID (see Figure 2.2). When a device is initialized, it can build local addresses for its interfaces by concatenating the link local prefix with a number that is unique to that interface. It was envisioned by the designers that network administrators would likely want to use the Media Access Control (MAC) layer physical addresses as this unique token, although it would not be strictly necessary. On 64 bits, a random number generator would create an address collision only about once in 300 million tries. Nonetheless, the physical address encoded in 48 bits in an IEEE-802 Ethernet card is very handy and guaranteed to be globally unique, at least if you don't use extremely cheap cards from unscrupulous manufacturers. A machine can trivially manufacture its own link-local scoped address of the form; FE80:0:0:0:xxxx:xxxx:xxxx:xxxx, where xxxx:xxxx:xxxx:xxxx represents the Ethernet physical address.

These link local addresses can only be used on the local link, which would be fine for a very small network without a router, but they are insufficient for organizing a large network.

### 2.7.2 Stateless autoconfiguration

Gai reviews the requirements of stateless autoconfiguration. Stateless autoconfiguration should not require any manual configuration or the presence of a stateful DHCP server. It must have the capability of generating unique link local

addresses automatically and using them for communications. Hosts must be able to derive site local or global addresses from router advertisements that contain lists of prefixes associated with links. Furthermore, stateless autoconfiguration should simplify renumbering operations and administrators should be able to choose between stateless and stateful configuration (or both). (Gai 1998)

Only multicast capable interfaces are able to autoconfigure themselves. When an interface is turned on or reset, it derives a link local address from the interface token. This address is not assigned immediately to the interface, but rather a duplication detection process is started. The host sends out neighbor solicitation packet on the network and listens for a reply. If there is a reply, then there is an interface with an identical address somewhere on the network, and a new address must be generated. Because packets may be dropped, the host may be required to repeat this process several times in order to be reasonably confident that the generated address is unique, although at present the default configuration is that the message is only sent once. It is only after this procedure that the address is assigned to the interface. The reasons for this precaution are that bad Ethernet cards do, in fact, exist and other tokens such as random number generation are allowed.

At this point the interface has a link local address. Both hosts and routers take this initial step, but only hosts perform the procedures that follow. The host must now obtain a router advertisement or verify that there are no routers on the network. The time interval between periodic router advertisements is fairly long, so the host may opt to send a router solicitation to the all-router (FF02::2) multicast address.

Recall that router advertisements contain to flags that indicate the kind of autoconfiguration to be performed (see Figure 2.14). If the M flag is set, the host must use stateful autoconfiguration for addresses. If the O flag is set, the host must use

stateful autoconfiguration for information other than addresses. Router advertisements also contain the address prefixes to be used in the stateless autoconfiguration of both site local and global addresses. Because these router advertisements are generated periodically, the addresses assigned to the host are continually updated and new addresses can be generated in response to new prefixes. As old addresses become invalid, the router no longer advertises them.

It should be noted that stateful and stateless autoconfiguration are not mutually exclusive. They can be used in parallel to configure both stateless derived addresses and stateful derived addresses.

There are security concerns with stateless configuration that may prompt some administrators to turn it off. The problem is that anyone who can bring a machine into the physical site of an organization and plug it in to a network jack can gain access to the network. It may be argued that this is also true of IPv4, but consideration should be given to authentication and encryption in order to provide security in the light of this particular vulnerability.

### 2.7.3 Stateful Autoconfiguration

The Dynamic Host Configuration Protocol version 6 (DHCPv6) is a protocol that was designed to provide IPv6 client nodes with configuration information that is stored on a server. The information provided by DHCP is mainly concerned with IPv6 addresses, but other information can be provided as well. The DHCP server manages the address and network parameters database. In a complex network, it is likely to be infeasible to have a DHCP server on each link, so the concepts of intermediary relays and agents have been introduced. A relay is a node that acts as an intermediary in the transmission of a packet between a client and a server, while an agent may be either a server or a relay.

DHCP uses the User Datagram Protocol (UDP) for communication, specifically port UDP 546 for agents to send messages and port UDP 547 for receiving all messages. There are six types of messages exchanged by the DHCPv6 protocol:

- (1) Solicit: these are messages sent to the multicast address of all DHCP server/relay agents (FF02::C). It is used when a client host doesn't know the address of a DHCP server.
- (2) Advertise: these are unicast messages sent to a client from a server in response to a solicit message.
- (3) Request: these are unicast messages from a client to a server requesting parameters for network configuration.
- (4) Reply: these are unicast messages from a server to a client in response to a request message. It contains the addresses and parameters that the server has allocated for the client.
- (5) Release: these are unicast messages from the client to inform the server that client has released previously allocated resources. The purpose of these messages is to allow the server to reallocate those resources to other nodes.
- (6) Reconfigure: these are unicast messages that are sent by the server to notify the client of modifications on the network. The client must get the specifics of the modifications through a request message and a server reply message.

It is worth noting that these messages are structured as request/reply pairs that constitute transactions between the client and the server. This structure is a result of the fact that UDP is an unreliable protocol, so the client and server must have a control mechanism. If a message packet is dropped or corrupted, the request message is resent until a valid reply message has been received. If the server needs to reconfigure the client, it requests the client to start a transaction through a DHCP reconfigure message.



#### 2.7.4 Site Renumbering

We have discussed the problems of a provider based addressing hierarchy at some length above. The process of renumbering an entire site as a result of a change of providers is daunting indeed. Besides the enormous quantity of sheer labor involved, the process is inordinately complex and errors are bound to occur. This process is much simplified by the mechanisms provided in IPv6.

In the present version 4 TCP/IP protocol suite, upper level protocols such as TCP identify connections, in part, by using the IP address. Thus, any change in the IP address will terminate all of the TCP connections that are in progress. The IPv6 designers have provided TCP with a mechanism for identifying addresses that will expire in the immediate future without actually mandating a change in the protocol. IP addresses have thus acquired a specifiable lifetime – they have become mortal.

To understand the IPv6 solution to this, we have to look at addresses from the upper level protocol point of view. There are two categories of addresses in IPv6: valid addresses and invalid addresses. Valid addresses are further subdivided into preferred addresses and deprecated addresses. When upper level protocols open a connection, they always use the preferred address.

When an administrator begins a renumbering procedure, he or she first inserts the new prefixes that will be used to build the new addresses into the routers and allows these prefixes to propagate throughout the network over a matter of several days. Then the prefixes of addresses that are no longer to be used are removed from the routers. This procedure creates new preferred addresses on all interfaces and turns some addresses that were preferred into currently deprecated addresses. These deprecated addresses are allowed to remain in the system for another few days to allow all TCP connections that were opened when the address was preferred to be closed gracefully.

The deprecated addresses are still valid, but they cannot be used for opening a new TCP connection. At last, the deprecated address becomes invalid and the transition is complete. Routers continue to announce both addresses during the transition.

#### 2.7.5 Address Resolution

As we have seen in section 2.3.3, address resolution is accomplished through ICMPv6 neighbor solicitations and neighbor advertisements. A node activates the procedure by multicasting a neighbor solicitation packet that requests the target node to return its MAC address. This neighbor solicitation packet is multicast to the solicited node multicast address associated with the target address. Starting with this multicast address, IPv6 algorithmically computes a multicast link layer address. This can happen in different ways depending on the type of link.

The target returns its link layer address in a unicast neighbor advertisement packet. This pair of messages is sufficient for both the initiator and the target to resolve each other's link layer address, since the initiator includes its link layer address in the neighbor solicitation message.

This neighbor solicitation/advertisement pair is the same one used in the duplicate address detection procedure of stateless autoconfiguration, and it will be used again in neighbor reachability detection, which is discussed in the next section

#### 2.7.6 Black Hole Detection

A hybrid topic that concerns both autoconfiguration and router discovery is neighbor unreachability detection. The results of the next hop computation are cached along with neighbor MAC addresses in most IP implementations. If this information becomes outdated, packets may be sent to a router that is no longer operational. As a result, they will disappear into the electronic void, into the Internet black hole. Thus

neighbor unreachability detection is defined as a part of the neighbor discovery mechanism.

A host can often learn that a destination is unreachable without any specific IP procedure, as when TCP fails to receive acknowledgement messages for any segments that it has sent to some address. IPv6 includes a procedure whereby TCP can pass reachability indications to the IP process to indicate that it is receiving acknowledgement messages as expected. If no reachability indications are received in a set time period (usually 30 seconds), the reachability of the destination becomes suspect.

The host may send a few more packets just in case, but after that a neighbor solicitation will be sent to ascertain that the destination is reachable. If no response is received this may be repeated a few times until the host decides that the neighbor has become unreachable and discards all packets bound for that destination. If the host receives a neighbor advertisement packet with the S (solicit) flag set from that destination, its reachability is confirmed.

Neighbor unreachability detection operates in parallel to packet transmission and is executed only in the presence of traffic.

For routers, the host must examine the R flag in the router advertisement message. If the host has an interface listed as a router in its cache, but its entry in the advertisement list lacks the R flag, that machine has ceased routing and packets should not be routed through it.

Flows like video packets over UDP or multicast will not generate reachability indicators and must be dealt with by other mechanisms such as the transmission of probes.

### III. THE TRANSITION STRATEGY FOR IPv6

#### 3.1 Overview

The developers of IPv6 were quite aware of the problems entailed in such an ambitious enterprise as upgrading the entire Internet to a new IP protocol. The solution that they devised was one of gradual deployment, with both protocols existing side by side for as long as it takes for the old IPv4 machines to expire of senescence. It must be said, however, that the change over has not been as rapid as the optimists had expected. Gai writes: "The years 1997 to 2000 will be characterized by the adoption of IPv6 by ISPs and users. During 1997, users could still have problems related to the newness of products, but starting from 1998, IPv6 will be part of mass-produced protocols distributed on routers, on workstations and on PCs. At that point, users will begin to migrate, less or more gradually, to IPv6." (Gai 1998)

Needless to say, this is not what has happened. The very process of implementing the code for the IPv6 and related protocol stacks and testing it for compliance with the protocol specifications has lasted well into this new millennium, an effort that continues, to some extent, to this day. Part of this may be due to Microsoft announcing delays to its support for IPv6 back in 1999. Furthermore, the urgency of the transition seems to have been lessened by the stopgap measures discussed in section 2.2.2 such as Classless Inter-Domain Routing (CIDR), Network Address Translation (NAT) and the Dynamic Host Configuration Protocol (DHCP), all of which contribute to a more efficient utilization of the available IPv4 address pool.

The underlying danger that the Internet will choke due to lack of address space and the computational overhead of grotesquely obese routing tables has not gone away. The implementations of the IPv6 stack for most operating systems are now reasonable

complete, mature and well tested. The United States Department of Defense has announced that all of its future software development contracts must be compliant with IPv6. Many large ISPs are maintaining an experimental presence on the 6Bone in order to gain experience with the new protocol, and Verizon, one of the US's largest telecommunication providers, has announced that it plans make native IPv6 network service available on a commercial basis in the near future. It is apparent that the transition is on the verge of really taking off.

There are three main transition strategies, all of which are aimed at allowing the two protocols to coexist on the same Internet as the migration to the new protocol proceeds. These strategies are: dual stacks, tunneling and header translation. Each will be discussed in its own section. Some IPv6 characteristics, such as the fact that IPv6 addresses can be automatically derived from IPv4 addresses, were explicitly designed to facilitate the transition. In addition, a set of mechanisms called Simple Network Transition (SIT) that consists of protocols and management rules to simplify the transition has been implemented.

SIT has as its stated goals, according to Gai:

- (1) The possibility of a progressive and non-traumatic transition: routers may be updated to IPv6 one at a time without requiring that other routers be updated at the same time.
- (2) Minimum requirements for updating: hosts will only require the availability of a DNS server that can hold IPv6 addresses. There are no special requirements for updating routers.
- (3) Addressing simplicity: routers and hosts that have been updated to IPv6 can still use IPv4 addresses.



- (4) Low initial costs: no preparatory work should be necessary to begin the transition to IPv6. (Gai 1998)

SIT has defined the following specific mechanisms in order to guarantee that IPv6 hosts can interoperate with IPv4 hosts initially anywhere on the Internet.

- (1) A structure of IPv6 addresses that allows the derivation of IPv6 addresses from IPv4 addresses.

- (2) Availability of a dual protocol stack on all hosts and routers during the transition.

A technique to encapsulate IPv6 packets within IPv4 packets to allow those packets to transverse regions of the Internet not yet updated to IPv6.

- (3) Ability for translation of IPv6 headers into IPv4 headers and vice versa to allow IPv4-only nodes to communicate with IPv6-only nodes. This is optional and will be used only in the advanced phases of the transition.

The purpose of this is to protect the current investment in IPv4 systems until they are retired from service due to obsolescence. This section will examine these strategies in some detail.

### **3.2 Dual Protocol Stacks**

Until all hosts on the Internet have completely migrated to IPv6, it is recommended that all Internet connected machines should run both IPv4 and IPv6 simultaneously. Since both IP and the OSI model visualize network software as a layered hierarchy or stack of functionality, the term dual stack is used to describe this strategy. Figure 3.1 schematically represents the dual stack structure.

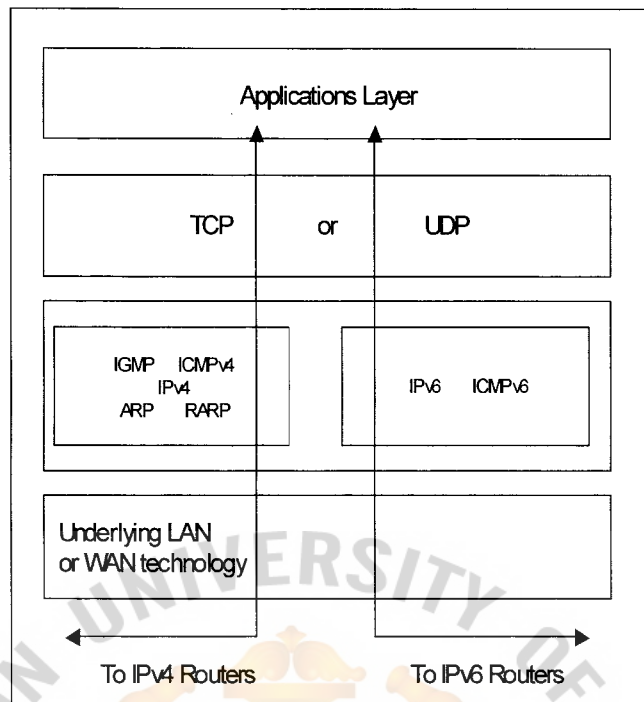


Figure 3.1. Dual Stack Schematic.

In a dual stack setup, the source host queries the Domain Name Server (DNS) for the address of the destination. If the destination's address is an IPv4 address, the host sends IPv4 packets. If the address returned is an IPv6 address, version 6 packets are transmitted. In the situation where the destination address is an IPv6 address with an embedded IPv4 address, IPv6 packets are encapsulated within IPv4 packets.

This can be a little tricky, as there are many more cases that are possible. RFC 2893 (cited in Gai 1998) contains a complete discussion of the handling of arcane situations. An example is the situation where both IPv4 and IPv6 addresses are stored in the DNS. It must be determined whether the node has direct IPv6 connectivity. If not the use of the IPv6 address will require the transmission of an IPv6 packet in an

IPv4 tunnel. This solution is likely to be less convenient than using native IPv4 packets, or may even be impossible if the destination node cannot use tunnels.

### 3.3 Tunneling

Tunneling is a nicely descriptive word for the practice of encapsulating packets of one protocol type into packets of another protocol type. This strategy is useful when two computers using IPv6 must send packets to each other through an IPv4 region of a network. To transverse such a region, the packets must have an IPv4 address, so the IPv6 packets are encapsulated into IPv4 packets as they enter the region, and may be unencapsulated when they leave the region, just as if they had entered a tunnel that passes through the region. The IPv4 packets that encapsulate the IPv6 packets are readily identified by their protocol value of 41.

There are two basic types of transition tunneling, automatic tunneling and configured tunneling. There are, of course, other uses for tunneling such as connecting intranets through the Internet as is done with virtual private networks (VPNs). As we shall see, the experimental IPv6 test bed backbone, the 6Bone, is based on tunneling techniques.

#### 3.3.1 Automatic Tunneling

When the destination machine has an IPv6 address in the DNS, tunneling occurs automatically and no further configuration is needed. The source transmits an IPv6 packet using the IPv6 address as the destination address. If the packet reaches the boundary of an IPv4 network, the router encapsulates it in an IPv4 packet using the IPv4 address of the destination host. The destination host's IPv4 address is embedded in the IPv6 address and so is readily derived. When the destination host receives the packet, it reads the IPv4 header and discovers through the protocol field value that the packet

contains an encapsulated IPv6 packet. This packet is passed to the IPv6 software for processing. Figure 3.2 illustrates this process.

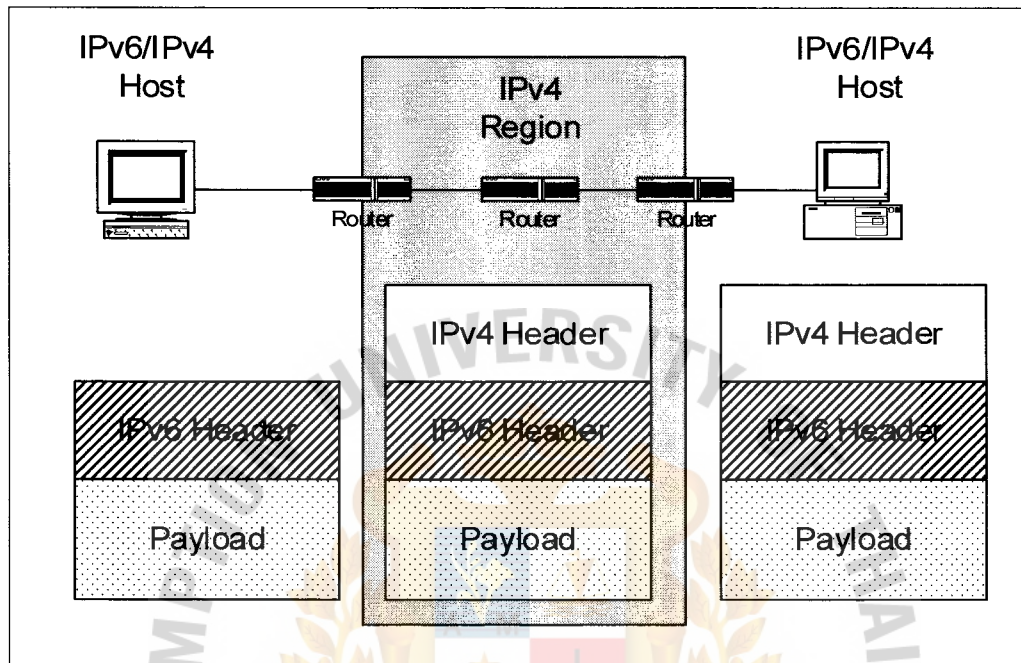


Figure 3.2. Automatic Tunneling.

Automatic tunneling is useful for host-to-host communication and for router-to-host communication where dual stack computers must transmit over IPv4 infrastructure.

### 3.3.2 Configured Tunneling

When a destination node does not support an IPv6 compatible address, the DNS returns a non-compatible IPv6 address to the sender. In this situation, the source node sends the IPv6 packet using that non-compatible address. When this packet must transit a IPv4-only region of the Internet, the router at the boundary of the region is configured to encapsulate the IPv6 in an IPv4 packet using its own address as the source address and the address of another router at the far end of the region as the destination address.

The second router decapsulates the packet and sends it on through the IPv6 region of the Internet where the destination host resides. The destination host, of course, processes the IPv6 packet normally. Figure 3.3 illustrates configured tunneling.

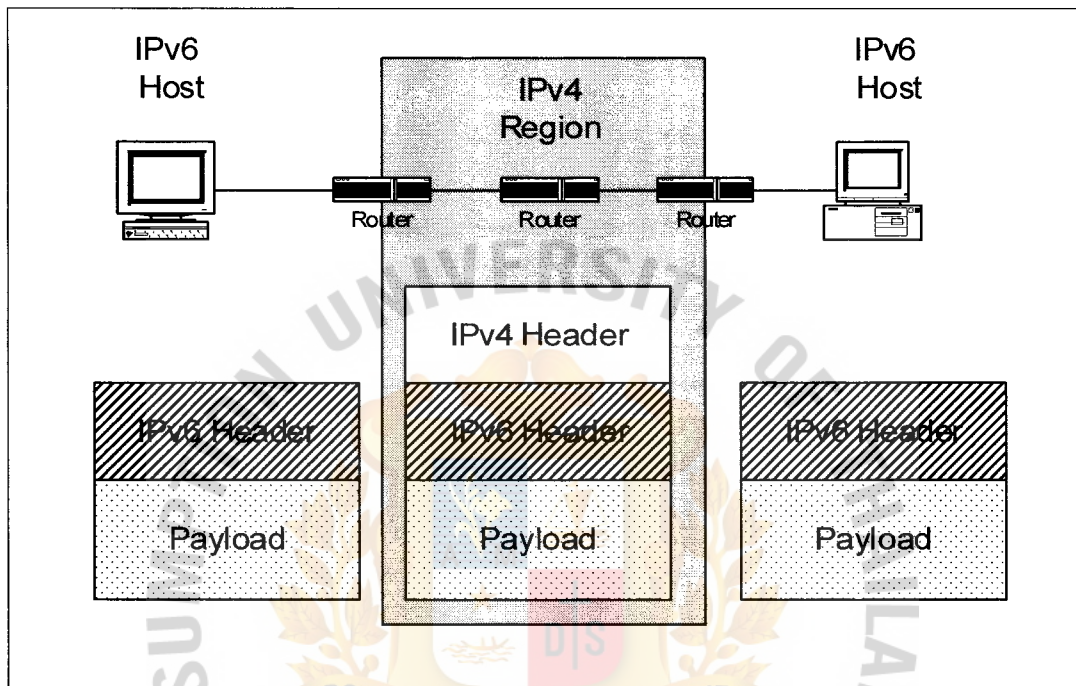


Figure 3.3. Configured Tunneling.

This type of tunneling is used for host-to-router and router-to-router communication since the router at the far end of the IPv4 region of the Internet must decode the IPv4 packet and forward it to its ultimate destination. There is no relationship between the address of the router and the final destination address. Therefore, the address of the tunnel end point router must be configured manually at the tunnel entry point.



### 3.4 Header Translation

Header translation will become necessary when the greater part of the Internet has completed the transition to IPv6. The relevant situation is when the source wants to use IPv6, but the destination can't understand it. Tunneling doesn't work here because the packet must be in the IPv4 format in order for the destination to understand it. In this situation, the IPv6 header must be translated into an IPv4 header and then reattached to the payload. Figure 3.5 illustrates this situation.

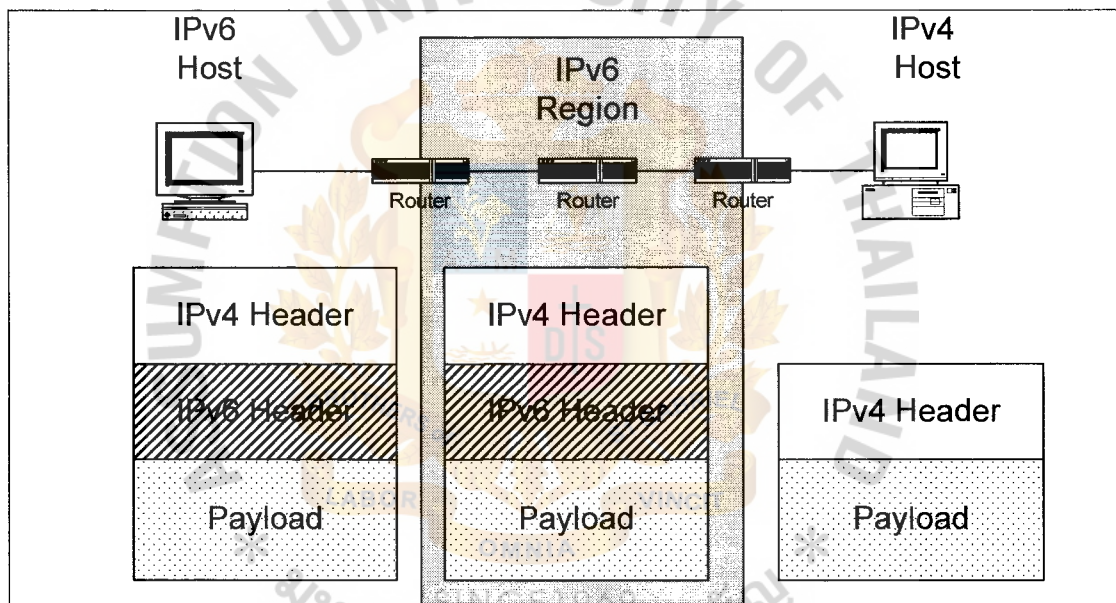


Figure 3.4. Header Translation.

Header translation uses the mapped address to translate an IPv6 address into an IPv4 address (See section 2.2.7). The following steps define the transformation of an IPv6 packet header into an IPv4 packet header:

- (1) The IPv6 address is changed to an IPv4 address by extracting the rightmost 32 bits.

- (2) The value of the IPv6 priority field is discarded.
- (3) The type of service field in IPv4 is set to 0.
- (4) The checksum for IPv4 header is calculated and inserted into the header checksum field
- (5) The IPv6 flow label is ignored.
- (6) Compatible extension headers are converted into options and inserted into the IPv4 options field.
- (7) The length of the IPv4 header is calculated and inserted into the length field.
- (8) The total length of the IPv4 packet is calculated and inserted into the total length field.

### 3.5 The 6Bone

At the time that the new protocol specifications were being implemented into actual code, some sort of test bed was thought necessary, initially for testing of standards and implementations and then later for testing transition and operational procedures. Although it turns out to be fairly easy to deploy on a stand-alone network, IPv6 is an Internet protocol and requires a real-world testing platform upon which to work out issues and give the software a rigorous workout.

The successful experience with the Mbone multicast test bed backbone, which was deployed in the spring of 1992, was fresh in the minds of the IETF engineers. A similar tunneling strategy, usually called 6to4 tunneling, was developed for the 6Bone, as was discussed in the previous section. The 6Bone was born during the IETF-Montreal meeting of 1996, and initially implemented in June and July of that year with tunnels between UL/PT, NRL/US and CISCO/US and between UNIC/DK, G6FR and WIDE/JP. How much it has grown is shown in Figure 3.5.

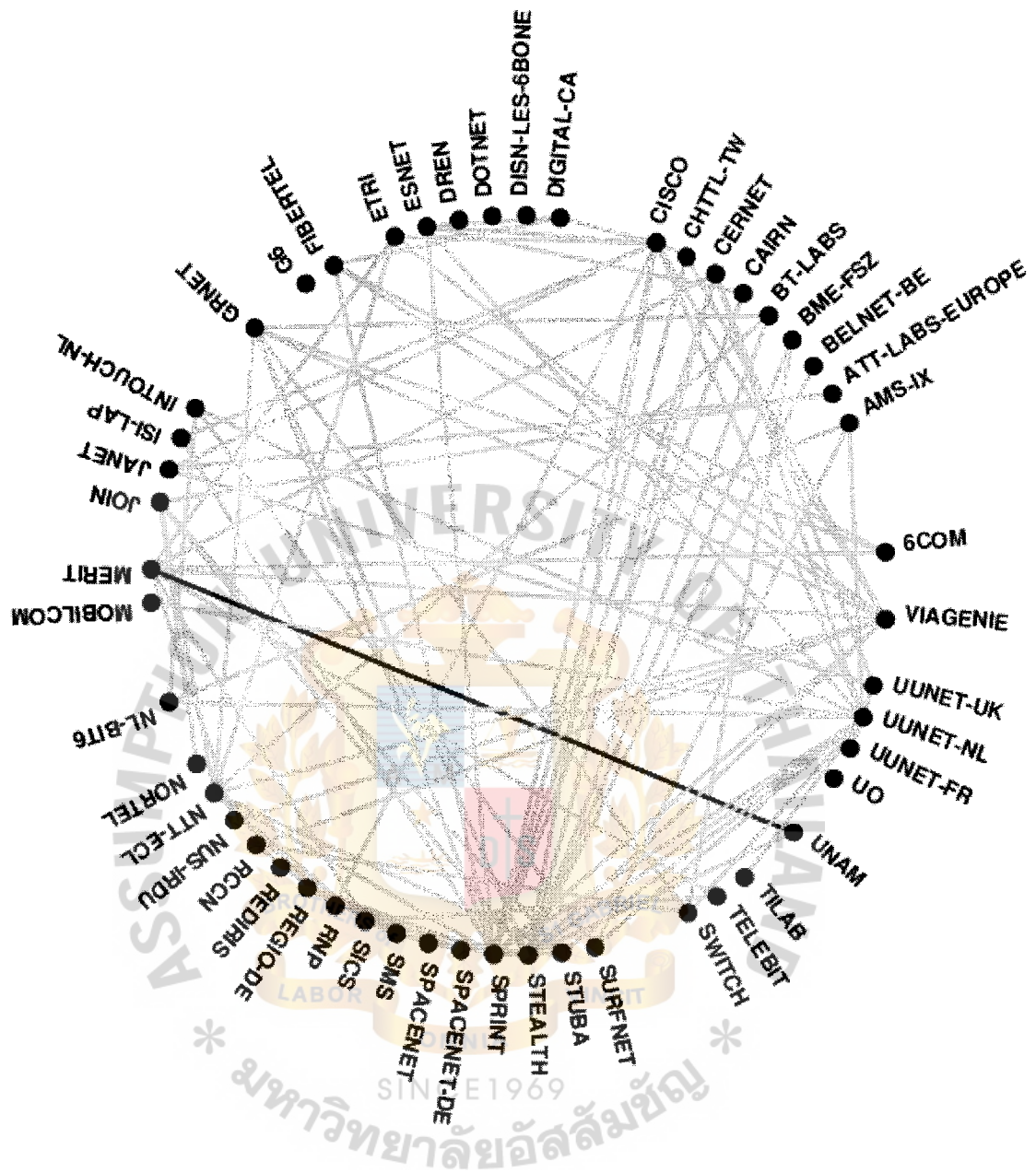


Figure 3.5. The 6Bone Backbone at Present.

In Thailand, the following organizations are currently maintaining a presence on the 6Bone: the Communications Authority of Thailand (CAT), Internet Thailand, Loxley Information Services, the National Electronics and Computer Technology Center (NECTEC) and Samart.

The 6Bone is an informal collaborative project that offers best-effort delivery for IPv6 over both tunnel links and native links. Because the purpose is strictly for testing, there are no service-level agreements among participating organizations. The 6Bone is conceived of as a time-limited project that will go out of business by common agreement as the production IPv6 Internet comes into service and becomes sufficiently extensive and trusted.

It is important to note that the 6Bone does not use the IPv6 production Internet address structures, but rather addresses allocated in RFC 1897 (Hinden and Postal 1996). These are considered test addresses, though any traffic to or from these addresses is (and will remain) valid without any limitation. These addresses start with the prefix 3FFE::/16 as compared to the 2001::/16 range given to non-test providers.

To connect to the 6Bone, a provider is required to connect to the service, but since these are very scarce at present, a convenient work-around has been established. The reserved IPv4 address 192.88.99.1 has been set aside for IPv6 tunneled traffic. The IPv4 backbone is configured so that any IPv4 traffic that is sent to this address will be routed to the nearest 6to4 relay. These relays have been constructed to decapsulate 6to4 packets and route them on the IPv6 backbone as well as performing the reverse process to packets on the return trip.

In order to accomplish this, a static IPv6 address is required. This will consist of the network prefix, which for 6to4 tunnels is 2002::/16, and the 112-bit host suffix. The first two blocks of four hexadecimal digits each are the unique global IPv4 address represented in hexadecimal. Local gateways usually are configured with the manual suffix ::1. As an example, the IPv4 address 209.81.9.15 produces the IPv6 address 2002:D151:90F::1.

Three kinds of 6to4 connections are possible. The first two handle the traffic between an IPv6 host and a 6to4 relay in either direction over IPv4 regions of the network. The third kind is between 6to4 IPv6 addresses. These connections are used within IPv6 clouds on the Internet so that a packet can pass from one 6to4 relay to another at the far end of the region where it is, in turn, once again encapsulated as an IPv4 packet for transmission to the destination over IPv4 links.

The fact that a static IPv4 address is required is a major problem with this scenario from the perspective of this project. Static IPv4 addresses are scarce, expensive and increasingly difficult to acquire now that NAT and DHCP are so widely deployed. Parenthetically, 6to4 tunneling will not work at all from behind a NAT server, or for that matter, from behind an IPv4 firewall. The next section discusses a solution to the problem of accessing IPv6 with dynamically assigned IPv4 addresses.

### **3.6 The Tunnel Setup Protocol and Freenet6**

Since there are as yet no commercial IPv6 service providers and it is so difficult to obtain static IPv4 addresses, users interested in experimenting with IPv6 developed a free and automated tunneling service that allows any individual or organization to connect with the IPv6 Internet. This service is named Freenet6.net. It is currently operated by Viagenie, a Quebec based consulting company, as a free, volunteer, best-effort based service.

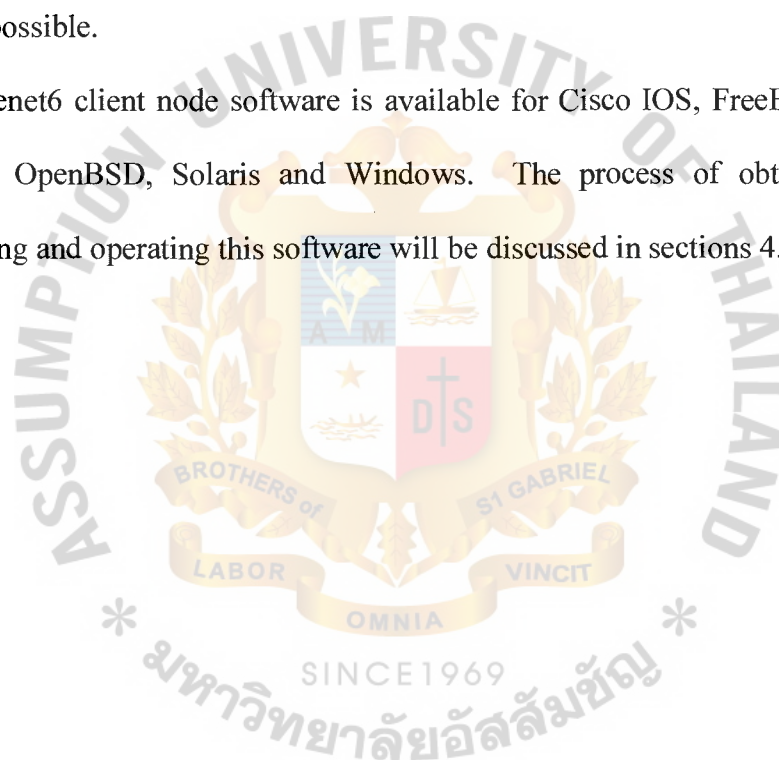
Freenet6 is software that is based on the tunnel broker concept set forth in RFC 3053. This software uses the Tunnel Setup Protocol (TSP) to negotiate the establishment of a tunnel between a client, which may be a host or a router, and the tunnel server. In addition to the tunneling service, the TSP server provides a large address space for routers, usually /48, which yields 16 bits of subnet addresses (up to 65,536 subnets), each of which may have up to  $2^{64}$  nodes (64 bits). This is astounding.



Just one Freenet6 router account offers more address space than the entire present-day Internet.

This software automatically handles the changes in the configuration of the tunnel whenever any of the endpoints of a connection change their IPv4 address. This is accomplished by the client-side software sending updated and authenticated updates to the server who maintains the account information. This relieves the problem of changing addresses due to DHCP assignment and makes dial-up access to the IPv6 Internet possible.

Freenet6 client node software is available for Cisco IOS, FreeBSD, GNU/Linux, NetBSD, OpenBSD, Solaris and Windows. The process of obtaining, installing, configuring and operating this software will be discussed in sections 4.5 and 4.6.



## IV. THE IPv6 WEB SERVER PROJECT

### 4.1 Requirements Analysis

The requirements for this system are extremely simple because this project is simply an exercise in making a system serve web pages using the IPv6 protocol suite. Formally, the system must meet the following requirements.

- (1) All necessary software must be installed. It may either be compiled from source code or installed from pre-compiled binaries.
- (2) The software must interoperate successfully. This implies that it should actually work using IPv6 protocol packets. This must be demonstrated using Ping, Traceroute and packet dump utilities.
- (3) IPv6 capable web server and web browser software must interact over the IPv6 experimental backbone (the 6Bone). These pieces of software must successfully serve and retrieve web pages respectively.

The necessary software will include an appropriate operating system with an IPv6 IP protocol stack, the Freenet6 utility appropriate to that operating system, an IPv6 enabled HTTP server, an IPv6 enabled web browser and a set of testing utilities that are compatible with the new protocol.

Ultimately, this project may be considered successful with regard to these requirements when the IPv6 website set up on one machine connected to the 6Bone over a dial-up connection serves web pages to another IPv6 enabled machine connected to the 6Bone over a separate dial-up connection.

## 4.2 The Existing System

The existing system is a small network of four computers that the writer uses for programming and experimentation. It might be thought of as a small computer laboratory for educational use. The specifications of the network capable computers in the existing system are listed in Table 4.1.

Table 4.1. Computer Hardware Specifications for the Existing System.

CPU	Memory	Hard Disk	Network Card	Comments
Intel Pentium III 1800 MHz	256 MB	40 MB	Ethernet 10/100 Allied Telesys	Built at Pantip Plaza
Intel Pentium III 460 MHz	256 MB	6 MB	Ethernet 10/100 Internal	Dell Optiplex GX1
DEC Alpha 166 MHz	128 MB	4 MB	Ethernet 10 DEChip 21040	
Intel Pentium III 330 MHz	96 MB	6 MB	Ethernet 10/100 PCMCIA	Toshiba Laptop PCMCIA Modem

Additional peripherals used in the existing system include:

- (1) Ethernet Switch: Surecom EP808x, 8 port.
- (2) Ethernet Hub: Surecom 505ST, 5 port.
- (3) External Modem: Lemel MD-56K.
- (4) KVM Switch: Jr. Super Commander, 4 station.
- (5) Ethernet Card: Billionton PCMCIA 10/100 Base
- (6) Ethernet Card: Allied Telesyn AT2500 10/100

The computers in the existing system are configured as disk partitioned dual-boot systems with the exception of the DEC machine, which can only run on GNU/Linux because no current Microsoft product supports the 64-bit Alpha processor. The dual-

boot configuration is supported by the Partition Magic partitioning utility software.

Table 4.2 shows the current systems operating system configuration.

Table 4.2. Operating System Configuration of the Current System.

Machine	First Operating System	Second Operating System
Intel Pentium III 1800 MHz	SuSE 8.2 Linux Professional Edition	Windows 2000 Professional
Intel Pentium III 460 MHz	BSD 4.4	Windows 98 SE
DEC Alpha 166 MHz	Red Hat Linux/Alpha 4.2	None
Intel Pentium III 330 MHz	SuSE Linux 7.1 Professional Edition	Windows 98 SE

These computers are all wired to the Ethernet switch using category 5 cables with RJ45 connectors. The network currently uses the IPv4 TCP/IP protocol suite. The Windows 2000 machine has an operational Internet Information Server (IIS) setup in Windows 2000 partition. The DEC Alpha machine has an operational Apache web server setup under Red Hat Linux. Neither of these server software suites is IPv6 compatible. The Netscape, Internet Explorer and Opera web browsers are available on all Windows partitions, while the Netscape, Opera, Mozilla, Galeon and Konqueror browsers are available on the GNU/Linux and BSD partitions. Of these, only the last IPv6 capable in the versions presently installed on the system. Connection to the external Internet is by way of the external modem, which may be attached to any of the machines, or by way of the PCMCIA modem in the laptop. Connection is through dial-up accounts held at KSC and Assumption University. There is one telephone line available in the lab room.

### **4.3 Platform and Software Selection for the Proposed System**

This section discusses the hardware available and the software choices that must be made for an experimental implementation of a web server over the IPv6 Internet. The hardware assets are fixed by what is available in the writer's small laboratory, but the software choices are considerable. A feasibility analysis, a cost analysis, a candidate system matrix and a discussion of the selection will be presented in the following sections.

#### **4.3.1 Hardware for the Proposed System**

The DEC Alpha machine is quite an antique and is only capable of running GNU/Linux. Unlike Intel machines, the Alpha uses a firmware bootstrap system that is very difficult to modify safely. The 1800 MHz machine from Pantip Plaza is a production machine that is needed for other critical uses. For these reasons, the Optiplex will be employed as the web server using the external 56K modem. Since there is only one telephone line installed in the laboratory room, the laptop will be configured as the client because it can be transported to another location in order to connect with the server over a separate PPP dial-up connection to the 6Bone. For initial site-local testing, the available Ethernet switch will be used. The architecture of the proposed system will be examined further after a discussion of software selection.

#### **4.3.2 Candidate Operating Systems for the Proposed System**

Three candidate operating systems will be considered: FreeBSD, GNU/Linux and Microsoft. These are all readily available in Thailand and are stable and mature. In order to compose a four-way candidate system matrix, two different IPv6 stacks will be considered for the GNU/Linux option. There is a version of Solaris available for the Intel x86 architecture, but it is excluded from consideration from the matrix for reasons of unfamiliarity. The operating systems themselves will be discussed first.



FreeBSD is an offshoot of the Berkeley Software Distribution of Unix. As the name implies, it is freely distributed and its source code is included with the binary distributions. Other members of the BSD family, such as NetBSD and Open BSD, are not considered because they are commercial products and rather expensive. At the outset of the project, FreeBSD is already installed on the Optiplex machine. This installed distribution, FreeBSD 4.4, was obtained, quite legally, from Pantip Plaza for about \$42, including a large reference book.

GNU/Linux needs little introduction. It is a freely available Unix clone, which, contrary to the current allegations of the SCO Group, does not employ proprietary Unix source code. The term Linux by itself refers only to the operating system kernel, so GNU/Linux is used in this paper to give credit to the Free Software Foundation's GNU project for all the many utilities and applications that they have made available to the world. As an operating system is much more than just the kernel, this nomenclature seems only just.

There are a large number of GNU/Linux distributions that vary in terms of the number and variety of software packages included, the installation scripts provided and the look and feel of the user interface. Although historically there were differences in the directory structure among distributions, a standard structure has been negotiated and is currently adhered to in general. The most well known distributions are Red Hat, SuSE and Debian. The writer has used the German SuSE distribution for a number of years because of its extensive provision of software packages and ease of installation. Red Hat is also a well-engineered distribution that is very popular in the US. Debian is a purist's distribution that is technically excellent, but contains no commercial or restricted software packages. Further discussion will limit the GNU/Linux option to be

one of the recent SuSE distributions because its solid Teutonic engineering has proven satisfactory in the past. SuSE Professional 8.2 was purchased previously for \$69 USD.

Microsoft Windows offers three products that can support an IPv6 stack: Windows 2000, Windows XP and Windows 2003 Server. Windows 2000, however, does not supply a native version 6 stack as it is distributed, and thus requires the downloading and compilation of a patch. As this would entail the additional expense of the purchase of compiler software, Windows 2000 is excluded from consideration. It is interesting to note, however, that this patch is delivered as source code, which is a breathtaking departure from the usual Microsoft policy.

Of the two remaining Microsoft operating systems, Windows 2003 Server is considered by Microsoft itself to be the product of choice and so will be the option to be considered here. The list price for Windows 2003 Server is \$999 USD for up to five processors. This is the cheapest package available, although street prices may be lower.

#### 4.3.3 Available IPv6 Protocol Stacks

Microsoft provides an IPv6 stack integral to its Windows 2003 Server product. This is no longer considered a beta product, although it is difficult to assess the relative maturity of the stack because no recent test results are available.

Open source IPv6 software is largely the product of a series of Japanese initiatives. The WIDE IPv6 working group was formed in 1995 for the purpose of experimenting with and deploying IPv6. As the specification was verified and interoperability became common, it was decided that the WIDE working group should focus on technical research, and a new implementation group, the KAME Project, was formed for implementation purposes. The KAME Project is a combined effort of a consortium of companies: Fujitsu, Hitachi, the I.I.J. Research Laboratory, NEC,

Toshiba and Yokogawa Electric. Their code is perhaps the most solid and mature code available. It is released under the BSD copyright.

For the purposes of conformance and interoperability testing, another group, the TAHI Project was also initiated in 1998. The TAHI group works closely with the WIDE working group and the KAME Project in providing verification technology and quality control. Yet another Japanese consortium that is closely associated with these efforts is the USAGI project, whose goal is to provide a quality IPv6 protocol stack for GNU/Linux based systems. Members include the WIDE Project, CRL, GLUON PARTNERS, INTEC, Toshiba, Hitachi, NTT, Yokogawa Electric, the University of Tokyo and Keio University. This project started in October of 2000.

The existing (version 2.4.5) Linux kernel also has its own native IPv6 protocol stack, but at that time, it was not considered as mature as the KAME stack or the Windows 2000 experimental stack (Haddad 2002) because of fluctuations in responsible personnel in the Linux development team. The USAGI stack is essentially a port of the KAME BSD stack to the GNU/Linux platform. Thus the open source stacks can be ordered as KAME, USAGI and the Linux kernel IPv6 stack in terms of stability, functionality and maturity. The deployment of the USAGI stack on GNU/Linux will require that the kernel source code be patched and recompiled.

The provenance of these pseudo-acronyms is of passing interest. In the Japanese language, KAME is the word for turtle or tortoise, and USAGI is the word for rabbit or hare. The word TAHI signifies a field day or track meet. Open source developers in Japan seem to be having an Aesopian field day indeed.

#### 4.3.4 HTTP Server Software

To some extent, the selection of web server software is dependent on the operating system chosen. For example, Microsoft Internet Information Server (IIS) runs only on

Microsoft operating systems, and is therefore not a consideration if an alternative operating system is chosen. The Apache web server software package is available for all of the hardware platforms under consideration. Other web server software does exist, but since IIS and Apache currently have a combined market share of approximately 90%, these other server products will not be considered. Should Windows 2003 server be chosen, IIS is bundled with the operating system. In order to install Apache on such a system, there are freely available binary builds of the software, but this would require a reasonably large download. Since IIS is native to the Windows environment, it would seem to be the web server software of choice for that platform.

Apache 1.3.8 comes bundled with the SuSE 8.2 distribution, both as source code ready for compilation using the free GNU compiler (GCC) and as a binary RPM (Red Hat Package Manager) package, but the level of support for IPv6 in this version is marginal. It is likely that a download and compilation of Apache2 would be required. The BSD option would require a download of the program source code, which is smaller than the binary code, but compilation would be required.

#### 4.3.5 Candidate System Feasibility Analysis

Four candidate systems will be considered for this project:

- (1) Candidate 1: Windows 2003 Server and IIS web server software
- (2) Candidate 2: SuSE Linux 8.2 Professional using the kernel IPv6 stack and Apache web server software
- (3) Candidate 3: SuSE Linux 8.2 Professional using the USAGI IPv6 stack and Apache web server software
- (4) Candidate 4: BSD 4.5 using the KAME IPv6 stack and Apache web server software.

These four choices seem to be representative of the range of choices available for the rather low-end server hardware that is available. It is perhaps regrettable that the Sun Solaris operating system is not proposed, but the personnel available for this experiment have no experience with Solaris whatsoever, so its feasibility is too questionable to justify inclusion. Table 4.3 gives the Candidate.

Table 4.3. Candidate Matrix for the Proposed System.

Characteristics	Candidate 1	Candidate 2	Candidate 3	Candidate 4
Benefits	Integrated commercial system. Installation support available.	Inexpensive. Integrated commercial system. Installation support available. IPv6 Stack already in the kernel. Familiar system.	Inexpensive. Integrated commercial system. Installation support available. Acceptable IPv6 stack. Familiar system.	Inexpensive. Most stable and mature IPv6 stack.
Disadvantages	Expensive. Lacks source code. Unfamiliar system.	Least stable and mature IPv6 stack	Must patch and recompile the kernel	Lacks commercial installation support. Unfamiliar system.
Hardware requirements	Existing hardware is sufficient.	Existing hardware is sufficient.	Existing hardware is sufficient.	Existing hardware is sufficient.
Software requirements	Windows 2003 Server Freenet6 download. Utilities package download.	SuSE 8.2 Professional Freenet6 download. Utilities package download.	SuSE 8.2 Professional USAGI kernel patch download. Freenet6 download. Utilities package download.	Free BSD 4.5 KAME IPv6 stack download. Apache server download. Freenet6 download. Utilities package download.



It should be noted that these benefits and disadvantages vary across several relevant factors including cost, availability of installation support, maturity and stability of the IPv6 stack, difficulty of stack installation and previous familiarity with the system. The lack of source code is a disadvantage in a system whose purpose is instructional.

Since cost is an extremely important factor, Table 4.6 lists the projected costs for each candidate system. The cost of already installed components is included.

Table 4.4. Projected Costs for the Candidate Systems (in US Dollars).

Item	Candidate 1	Candidate 2	Candidate 3	Candidate 4
Dell OmniPlex Computer (used)	145	145	145	145
Toshiba Laptop Computer (used)	525	525	525	525
Surecom Ethernet Switch	56	56	56	56
Billionton Ethernet Card	40	40	40	40
Allied Telesyn Ethernet Card	29	29	29	29
Ethernet Cat5 Cables	4	4	4	4
Lemel External Modem	59	59	59	59
Apollo PCMCIA Modem	39	39	39	39
Operating System	999	69	69	42
Total	1896	966	966	939

The only new cost is \$999 USD for the operating system for Candidate System 1. All other components have been long since bought and paid for. All other necessary software is either included in the operating system packages or is free for downloading.

The feasibility of the candidate systems will be considered in terms of four feasibility criteria: Operational Feasibility, Economic Feasibility, Technical Feasibility and Schedule Feasibility. These criteria are weighted as follows:

- (1) Operational Feasibility is factored at 10%. There is no plan to make this system into an operational production system.
- (2) Economic Feasibility is factored at 40%. Funds for this project are extremely limited.
- (3) Technical Feasibility is factored at 20%. It is believed that any of these candidate systems can attain the minimal operability specified in the requirements.
- (4) Schedule Feasibility is factored at 30%. There are time constraints on the completion of this project that cannot be ignored.

In order to provide support for a reasonable choice among the candidate solutions, it is useful to examine a Feasibility Matrix. Table 4.5 presents a Feasibility Matrix for these candidate systems.

Table 4.5. Feasibility Matrix for the Candidate Systems.

Feasibility Criterion	Weight	Candidate 1	Candidate 2	Candidate 3	Candidate 4
Operational Feasibility	10%	Operational requirements are likely to be met. The current level of IPv6 operability is something of a mystery.  Score: 80	Operational requirements are likely to be met. There is some concern about whether the kernel IPv6 stack will support this sufficiently.  Score: 60	All operational requirements are met.  Score: 100	All operational requirements are met.  Score: 100
Economic Feasibility	40%	The Microsoft 2003 Server operating system is very expensive. A budget might be found for it, but not if an alternative exists.  Score: 10	Not the most inexpensive alternative, but still very economical. This software has already been purchased in any case.  Score: 97	Not the most inexpensive alternative, but still very economical. This software has already been purchased in any case.  Score: 97	The most economical of the candidates. This software has already been purchased in any case.  Score: 100

Table 4.5. Feasibility Matrix for the Candidate Systems (continued).

Feasibility Criterion	Weight	Candidate 1	Candidate 2	Candidate 3	Candidate 4
Technical Feasibility	20%	<p>It is difficult to ascertain the level of technical proficiency of Microsoft systems. As this software is no longer in beta testing, we can give them a cautious benefit of the doubt.</p> <p>Score: 80</p>	<p>This is most certainly the least stable and mature IPv6 stack. It might manage to meet the requirements but there is no guarantee.</p> <p>Score: 60</p>	<p>This IPv6 stack is ported from the KAME stack for BSD. It may not have the same level of code maturity and testing.</p> <p>Score: 90</p>	<p>The KAME stack is the state-of-the-art. Stable, reliable and mature.</p> <p>Score: 100</p>
Schedule Feasibility	30%	<p>No particular problems are foreseen, but this product is not familiar to the person implementing the project. Installation support is available.</p> <p>Score: 90</p>	<p>This candidate is very familiar to the person implementing the project. Installation support is available.</p> <p>Score: 100</p>	<p>This candidate is very familiar to the person implementing the project. Installation support is available.</p> <p>Score: 100</p>	<p>Although this candidate is a Unix variety, it is not very familiar to the person implementing the project. Details of compilation and installation will have to be learned</p> <p>Score: 85</p>
Ranking	100%	55	86.8	96.8	95.5

Thus we see that the SuSE Linux using the USAGI stack narrowly edges out the BSD with the KAME stack option. This decision is forced by the tight timeline that the project must follow to achieve schedule feasibility. Nonetheless, such a compromise should result in a successful implementation. The project will be implemented using the SuSE 8.2 Professional edition on the server. The server's new 2.4.20 kernel will be recompiled with the latest USAGI IPv6 stack patch. The client is a very small laptop that lacks a CD-ROM drive and does not even have a parallel port. As such, software installation is rather a delicate issue for this machine. However, since the source code for the 2.2.7 Linux kernel is already on the machine, it is proposed that the USAGI patch be applied to that kernel in lieu of a complete reinstallation. For reasons that will be discussed in section 5, Opera version 7.23 was the browser originally chosen for installation on the client, but circumstances dictated that Mozilla 1.2.1 be used during the actual installation.

#### **4.4 Design of the Proposed System**

With the hardware and software platforms decided upon, design of the system can proceed. Figure 4.1 depicts the hardware architecture for the proposed system, which will prove useful in the subsequent discussion of the software architecture.



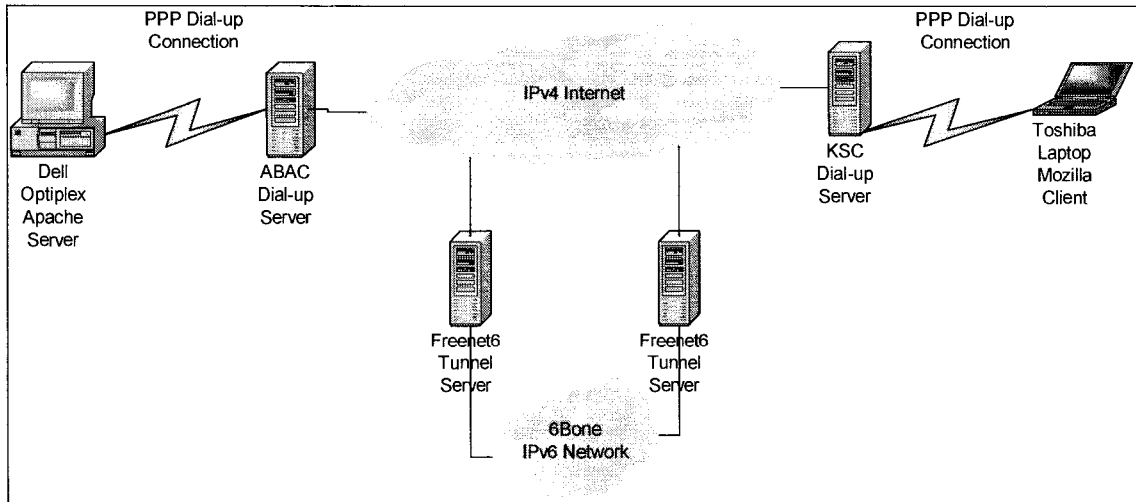


Figure 4.1. Architecture of the Proposed System.

The Dell Optiplex machine will have the following software configuration:

- (1) SuSE Linux 8.2 Professional kernel recompiled with the USAGI IPv6 stack.
- (2) Apache 4.5 HTTP server software.
- (3) Freenet6 software utility software configured for the server.

This configuration will allow the server to produce IPv6 address compliant HTTP responses, which are then converted into native IPv6 packets by the kernel. The Freenet6 utility manages the encapsulation of the IPv6 packets into IPv4 packets as well as the account information necessary for the Freenet6 server to manage the association between the static IPv6 address and the dynamic IPv4 address of the server.

The encapsulated packets are then sent to the ABAC dial-up server via a dial-up PPP connection through the Lemel 56K modem. The IPv4 header address of the encapsulated IPv6 packets is that of the Freenet6 server. These packets are routed through the IPv4 cloud and are decapsulated at the Freenet6 server. As mentioned above, this server maintains a table that associates the account information of the Freenet6 client machines with their dynamic IPv4 addresses and their static IPv6

addresses. The 6Bone routing architecture is such that these static IPv6 addresses are automatically routed through the Freenet6 server system for further IPv4 tunneling processing. The project web server and the project web client will maintain two different Freenet6 client accounts.

The packets are then routed through the 6Bone cloud, at least theoretically. It is entirely possible that the Freenet6 server will recognize the destination address in its own routing table. However, since IPv6 decapsulation has taken place and re-encapsulation follows, this is not considered detrimental to the requirements of the project.

After being processed through the IPv6 portion of this journey, the re-encapsulated packets pass back through the IPv4 portion of the Internet to the KFC dial-up server that is connected via a separate PPP dial-up connection to the PCMCIA modem on the Toshiba laptop. This machine uses its own Freenet6 utility software configured for its own separate Freenet6 account. The packets are processed through a SuSE Linux/USAGI protocol stack homologous to that in the web server machine, but compiled separately so that it is properly configured for the different hardware. The packets are then handed off to an IPv6 compliant Mozilla browser.

In addition, both machines will have three suites of testing utilities installed: the net-tools collection, the iputils utilities and NetKit. The most important of these tools are ping6, traceroute6 and tcpdump for debugging and exploration of the system.

After implementation of these software and hardware platforms and applications, the plan is to initially test the software over the installed 10/100 Ethernet LAN. If successful, the server machine will dial out to its ABAC account and establish a connection with the Freenet6 server. The client laptop will be taken to another location and a connection to its Freenet6 server will be established through the KSC dial-up

server. The client will then issue an HTTP request to the IPv6 address of the project server. It is hoped that the server will receive this request and send a response through the paths specified above.

#### **4.5 Implementation of the System**

This section describes in rather fine-grained detail the process of gathering software packages, uncompressing archives and, in some cases, compiling software in a Unix environment.

The commands discussed below were entered in either full screen console terminal mode or into a console window in the X Window graphical user interface using one of several window managers including KDE and Gnome. The default shell interpreter in GNU/Linux systems is Bash (the Bourne Again Shell), which is an extension of the original Unix Bourne shell. Other shells are available, notably the C, Z and Tcl shells, but bash is full-featured and reasonably easy to use. It is a revelation to anyone whose shell programming experience is limited to MS DOS command.com. The frequent use of the “./” prefix before a command in these examples is a mechanism that forces the shell to include the current directory in the path.

It goes without saying that all these commands must be executed with root privileges, as only the superuser should be allowed to delve so deeply in the internals of the system.

##### **4.5.1 Software Installation on the Server**

Implementation of the system began with an installation of SuSE Linux 8.2 on the Optiplex machine. This went very smoothly with no technical problems. However, one of the difficulties of working with an open source product is that there are not necessarily good drivers for all hardware. This is especially problematic for those Ethernet Network Interface Cards that are integrated on the motherboard. The one in

the Optiplex was recognized by the SuSE installation software, but the driver installed did not allow communication on the network. Therefore, an Allied Telesys card was scavenged from another machine and installed on the Optiplex. It was automatically recognized by the operating system and the correct driver was installed, again automatically. The results were quite satisfactory for Ping utility, and with the proprietary YaST (Yet another System Tool) configuration tool, the Internet services daemon inetd was properly configured to allow FTP and Telnet to other machines on the Ethernet network.

A copy of the USAGI 2.4.19 RPM was obtained from the site listed in Appendix A. The directory location was `pub/usagi/stable/package/SuSE/RPMS/20021007/`. This download was quite large, amounting to almost 14 megabytes. It was decided to go with the RPM package rather than compiling from source code, as the source packages were two to three times larger.

The RPM package was installed on the server machine quite simply by typing `"rpm -i k_usagi-2.4.19.usagi-20021007.i386.rpm"` at the command line. This was not actually very demanding because the long filename can be completed in the bash shell interpreter with the tab key. One major benefit of using RPM binaries in GNU/Linux is that they are automatically checked for library dependencies. In addition, they are much faster to download and install.

The net-tools, iputils and NetKit utilities packages were also downloaded from their respective websites (see Appendix A). The net-tools package was a tar.bz2 archive, which was unzipped using the `"tar xlvf net-tools-1.60.tar.bz2"` command. Then the pre-compilation configuration program was invoked with `"./configure.sh config.in"`. This configuration utility asked several questions, and the defaults were accepted with three exceptions: INET6 protocol family, SIT support and build iptunnel and ipmaddr

were changed to yes. Then “make && make install” started the compilation and installation of the package.

The iputils package came as a tar.gz file inside of an RPM package. After running RPM, the archive was unpacked using the “tar -xzf iputils-ss001110.tar.gz” command. This package does not have an install utility, so the “make” command by itself invoked the compiler, and the binary was manually installed in the directory /usr/local/iputils/bin.

The NetKit package was a tar.gz archive, which was unpacked with the command, “tar -xzf nkit-0.5.1.tar.gz”. The configuration utility was invoked with “./configure”, followed by “make” and “make clean”. The resultant binaries were moved to usr/local/bin.

The latest stable Apache2 server source code (version 2.2.4) was obtained from the Apache website as another large download of just over 6 megabytes. It was unzipped using the “gzip -d httpd-2.0.48.tar.gz” command followed by the “tar -xvf httpd-2.0.48.tar” command. The configuration utility was invoked with “./configure”, followed by “make” and “make install”. The configuration installed the files into the /usr/local/apache2/ directory.

These compilations and installations all proceeded normally, and the resulting software was operational. This was not always the case, as we shall see in the next section. The testing of the installed server software will be discussed in section 4.6.

#### 4.5.2 Software Installation on the Client

As was discussed in section 4.4, the Toshiba laptop is an ultra-light machine that lacks a CD-ROM drive and even a fixed parallel port. The original GNU/Linux installation on this machine required the painstaking process of booting into DOS, loading of the contents of each installation CD into a dedicated partition of the hard



drive and rebooting into the SuSE installer for each of the five installation CDs. In order to avoid a repetition of this, it was decided to keep the present installation, but to compile version 2.4.20 of the Linux kernel to replace version 2.2.4 and use that for the experiment.

The source code files of recent Linux kernels are very large, so it was decided to use the source code file in the SuSE 8.2 distribution CDs. A Network File System (NFS), with the Optiplex as the server system and the CD-ROM drive mounted as a share, was set up to handle the transfer of this large source code file. The file was then moved into the directory `/usr/src/linux-2.4.20` and a symbolic link was made with the command `ln -s /usr/src/linux-2.4.20 linux.` At this point a boot floppy was made as insurance against an unbootable system. In addition, the actual bootable kernel was copied to a backup with the command `cp vmlinuz vmlinuz.old`, and the following lines were added to the boot loader configuration file, `/etc/lilo.conf`:

```
image = /boot/vmlinuz
root = /dev/hda7
label = old
```

This addition causes an option to be presented by the boot loader to boot the original kernel image. This is necessary because the process of compiling and installing a new kernel overwrites the original kernel.

The kernel was then custom configured using the `./xconfigure` command, which invokes a graphical menu system for the custom design of kernel components to be compiled into the new system kernel. This utility offers very fine-grained control over both internal kernel features and modules that can be dynamically linked to the kernel at run time. The output of this utility is a configure file that is a long list of C language `#define` statements. The actual compilation process requires the regeneration of all the dependency files introduced by the options that were configured. Thus, the first

compilation command is “make dep clean”. This can be chained with the compilation, module linkage and installation commands as follows: “bzlilo modules modules\_install”. These are commands that are defined in the makefile, which the Make utility refers to in order to issue compiler commands with the proper arguments.

After issuing this command, the console will display a seemingly endless series of compiler commands, reports and warnings, which can last for a very long time depending on the processor speed and memory of the hardware. Anyone replicating this installation needs to be aware of this and not to interfere with the process until the system once again displays a command prompt.

This compilation completed successfully and the new kernel booted uneventfully. The USAGI patch was applied against the new kernel using the steps outlined for the server in section 4.5.1.

The problem of installing an IPv6 enabled web browser proved to be quite vexatious. The browsers on the current system could not parse the long IPv6 addresses. Source code for the Mozilla 1.3 browser was located and a compilation was attempted. This compilation was a failure because it required a newer set of C and Qt (the TrollTech Windowing Toolkit) libraries that were available on the machine. This is a common problem in open source development because the software libraries are constantly being extended and improved. This writer has worked around this problem on occasion by symbolically linking the filenames required by the makefile to old library modules, but this is precarious and often results in unstable or unpredictable programs. It was felt to be easier to find a browser package that was already compiled with statically linked Qt libraries than to track down and install the missing libraries.

The Opera browser version 7.19 has a Qt static Linux distribution at their website (see Appendix A), which was downloaded and installed via the command: “rpm -i

opera-7.23-20031119.1-static-qt.i386-en.rpm". Once again, Bash filename completion was appreciated. This browser installation was successful.

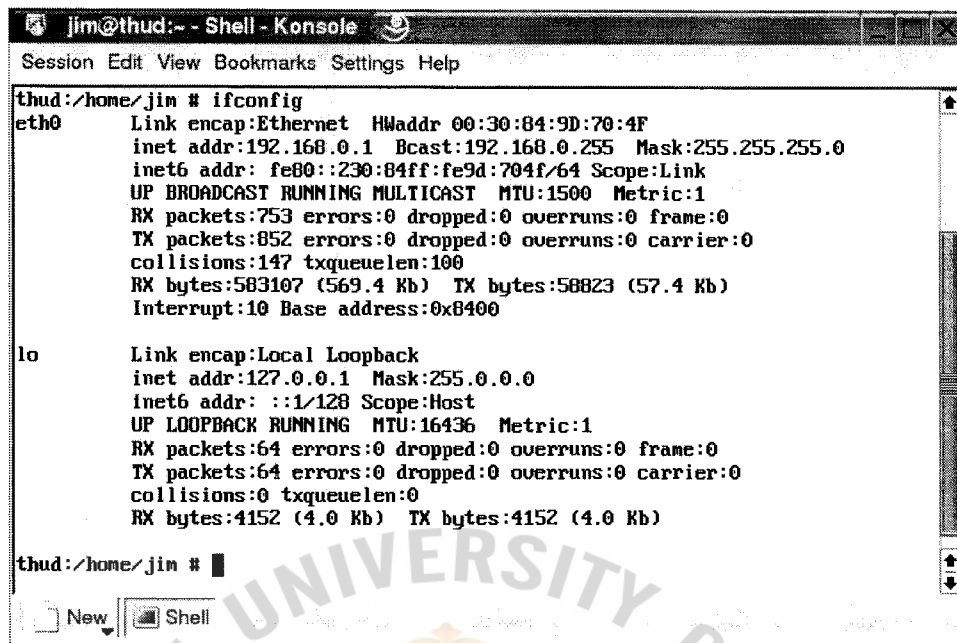
The Freenet6 client software was obtained as an RPM package. It was installed on the server with no problems, but the client had four missing libraries. At this juncture, it seemed more expeditious to simply install SuSE Linux 8.2 on the laptop than to track down the missing libraries by hand. This was accomplished with some effort and the Freenet6 software was installed successfully into the /usr/local/freenet6-client directory. Two accounts were opened at the Freenet6 site, and the automatically assigned passwords were emailed to the writer. In passing, it is important to note that two separate email accounts are necessary, as only one account can be allocated per email address. The usernames and passwords were cut and pasted into the file /usr/local/freenet6-client/bin/tsp.conf in both the server and the client machines.

At this point the system was ready for initial testing over an Ethernet network, as will be reported in the next section.

## **4.6 Testing the System**

### **4.6.1 Testing the System on an Ethernet LAN**

As mentioned in section 4.5.1, IPv4 capabilities were tested with Ping, FTP and Telnet, and a Network File System was configured. In order to discover the link local IPv6 addresses on the two machines, the "ifconfig" command was run. This command displays the configuration of the network interfaces. The results of these dumps are shown in Figures 4.2 and 4.3 for the server and client respectively.



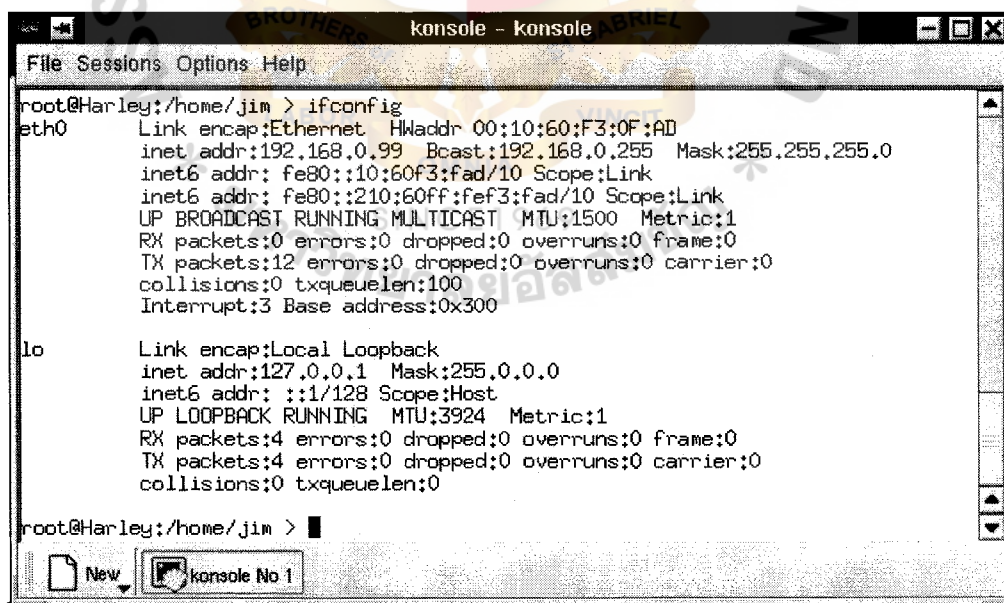
```
jim@thud:~ - Shell - Konsole
Session Edit View Bookmarks Settings Help

thud:/home/jim # ifconfig
eth0      Link encap:Ethernet  HWaddr 00:30:84:9D:70:4F
          inet addr:192.168.0.1  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::230:84ff:fe9d:704f/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:753 errors:0 dropped:0 overruns:0 frame:0
          TX packets:852 errors:0 dropped:0 overruns:0 carrier:0
          collisions:147 txqueuelen:100
          RX bytes:583107 (569.4 Kb)  TX bytes:58823 (57.4 Kb)
          Interrupt:10 Base address:0x8400

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:64 errors:0 dropped:0 overruns:0 frame:0
          TX packets:64 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:4152 (4.0 Kb)  TX bytes:4152 (4.0 Kb)

thud:/home/jim #
```

Figure 4.2. Server Interface Configuration.



```
konsole - konsole
File Sessions Options Help

root@Harley:/home/jim > ifconfig
eth0      Link encap:Ethernet  HWaddr 00:10:60:F3:0F:AD
          inet addr:192.168.0.99  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::10:60f3:fad/10 Scope:Link
          inet6 addr: fe80::210:60ff:fef3:fad/10 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:12 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          Interrupt:3 Base address:0x300

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:3924  Metric:1
          RX packets:4 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0

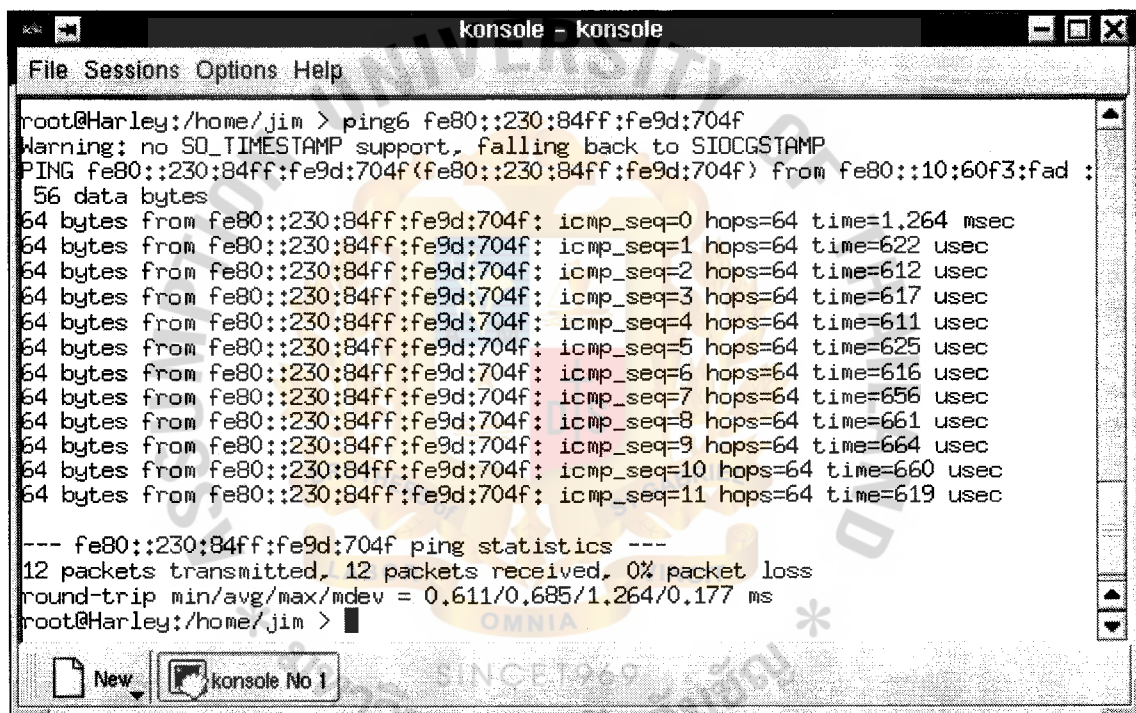
root@Harley:/home/jim >
```

Figure 4.3. Client Interface Configuration.



The server was configured with the IPv6 address FE80::230:84FF:FE9D:704F/64 for the eth0 interface, while the client configured itself with two IPv6 addresses for its eth0 interface. The address that it uses in practice is FE80::10:60F3:FAD.

Initial testing of the IPv6 connectivity was performed from the client to the server using the “ping 6” command along with the link local IPv6 address. The result of this command with the manually entered IPv6 address is shown in Figure 2.4:



```
konsole - konsole
File Sessions Options Help
root@Harley:/home/jim > ping6 fe80::230:84ff:fe9d:704f
Warning: no SO_TIMESTAMP support, falling back to SIOCGSTAMP
PING fe80::230:84ff:fe9d:704f(fe80::230:84ff:fe9d:704f) from fe80::10:60f3:fad :
 56 data bytes
64 bytes from fe80::230:84ff:fe9d:704f: icmp_seq=0 hops=64 time=1.264 msec
64 bytes from fe80::230:84ff:fe9d:704f: icmp_seq=1 hops=64 time=622 usec
64 bytes from fe80::230:84ff:fe9d:704f: icmp_seq=2 hops=64 time=612 usec
64 bytes from fe80::230:84ff:fe9d:704f: icmp_seq=3 hops=64 time=617 usec
64 bytes from fe80::230:84ff:fe9d:704f: icmp_seq=4 hops=64 time=611 usec
64 bytes from fe80::230:84ff:fe9d:704f: icmp_seq=5 hops=64 time=625 usec
64 bytes from fe80::230:84ff:fe9d:704f: icmp_seq=6 hops=64 time=616 usec
64 bytes from fe80::230:84ff:fe9d:704f: icmp_seq=7 hops=64 time=656 usec
64 bytes from fe80::230:84ff:fe9d:704f: icmp_seq=8 hops=64 time=661 usec
64 bytes from fe80::230:84ff:fe9d:704f: icmp_seq=9 hops=64 time=664 usec
64 bytes from fe80::230:84ff:fe9d:704f: icmp_seq=10 hops=64 time=660 usec
64 bytes from fe80::230:84ff:fe9d:704f: icmp_seq=11 hops=64 time=619 usec

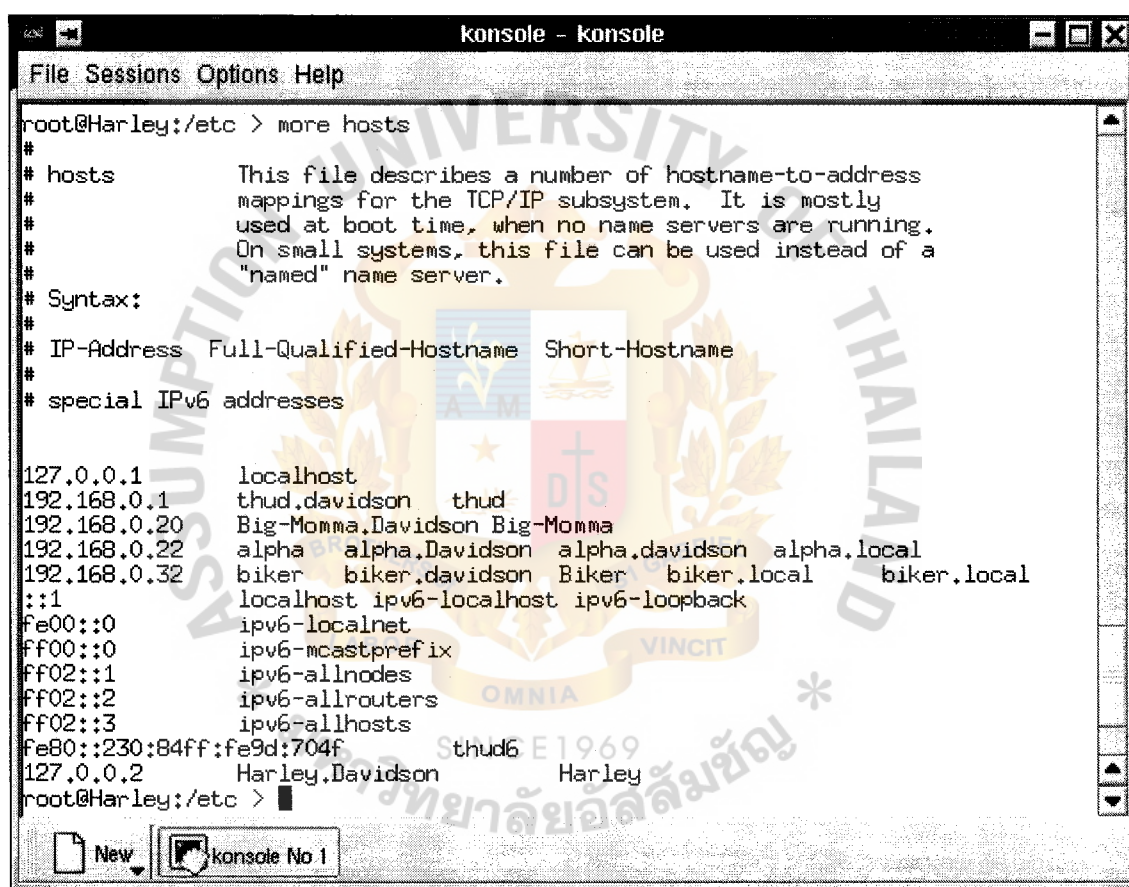
--- fe80::230:84ff:fe9d:704f ping statistics ---
12 packets transmitted, 12 packets received, 0% packet loss
round-trip min/avg/max/mdev = 0.611/0.685/1.264/0.177 ms
root@Harley:/home/jim >
```

Figure 4.4. Connectivity Test from Client to Server.

The Tcpdump utility was used to capture a dump of all the IPv4 packet headers during a Ping session for comparison of a similar dump of IPv6 packet headers during a ping session. These are of interest because they illustrate the packet header contents that were discussed extensively in section 2.3. The dumps are found in Appendix B.



Needless to say, typing in a full IPv6 address is laborious. Since there is no DNS server configured at this point, these addresses were aliased to a host name in the /etc/hosts files of the two machines. In addition, as we shall see later, these aliases can be used in the browser address window. The /etc/hosts file of the client computer is shown in Figure 2.5.



```

konsole - konsole
File Sessions Options Help
root@Harley:/etc > more hosts
#
# hosts          This file describes a number of hostname-to-address
#                mappings for the TCP/IP subsystem.  It is mostly
#                used at boot time, when no name servers are running.
#                On small systems, this file can be used instead of a
#                "named" name server.
# Syntax:
#
# IP-Address     Full-Qualified-Hostname  Short-Hostname
#
# special IPv6 addresses
#
127.0.0.1        localhost
192.168.0.1      thud.davidson  thud
192.168.0.20     Big-Momma.Davidson Big-Momma
192.168.0.22     alpha.alpha.Davidson alpha.davidson alpha.local
192.168.0.32     biker.biker.davidson Biker biker.local biker.local
::1             localhost ipv6-localhost ipv6-loopback
fe00::0         ipv6-localnet
ff00::0         ipv6-mcastprefix
ff02::1         ipv6-allnodes
ff02::2         ipv6-allrouters
ff02::3         ipv6-allhosts
fe80::230:84ff:fe9d:704f thud6
127.0.0.2       Harley.Davidson Harley
root@Harley:/etc >

```

Figure 4.5. Aliases in the etc/hosts File.

Using these aliases in the etc/hosts file, Figure 4.6 shows a Ping6 interaction that is much easier to type.

```

konsole - konsole
File Sessions Options Help
root@Harley:/etc > ping6 thud6
Warning: no SO_TIMESTAMP support, falling back to SIOCGSTAMP
PING thud6(thud6) 56 data bytes
64 bytes from thud6: icmp_seq=0 hops=64 time=645 usec
64 bytes from thud6: icmp_seq=1 hops=64 time=607 usec
64 bytes from thud6: icmp_seq=2 hops=64 time=600 usec
64 bytes from thud6: icmp_seq=3 hops=64 time=602 usec
64 bytes from thud6: icmp_seq=4 hops=64 time=612 usec
64 bytes from thud6: icmp_seq=5 hops=64 time=601 usec
64 bytes from thud6: icmp_seq=6 hops=64 time=600 usec
64 bytes from thud6: icmp_seq=7 hops=64 time=601 usec
64 bytes from thud6: icmp_seq=8 hops=64 time=600 usec
64 bytes from thud6: icmp_seq=9 hops=64 time=602 usec
64 bytes from thud6: icmp_seq=10 hops=64 time=599 usec
64 bytes from thud6: icmp_seq=11 hops=64 time=612 usec

--- thud6 ping statistics ---
12 packets transmitted, 12 packets received, 0% packet loss
round-trip min/avg/max/mdev = 0.599/0.606/0.645/0.032 ms
root@Harley:/etc >

```

Figure 4.6. Ping with Aliased IPv6 Address.

The next testing step was to try the Apache2 server on the loopback address. It should be noted here that the SuSE Linux installation program installed Apache 1.3.27 as the default web server, and that this configuration cannot be removed without altering dependencies. Since the server does not run at start up, this can be overcome by moving to the `/usr/local/apache2/bin/` directory and typing `./apachectl start`. Using the Mozilla 1.2.1 browser that came with the distribution, the server was first tested for its IPv4 address and the aliased local host value of `ipv6-localhost`. The result of the latter test is shown in Figure 4.7.

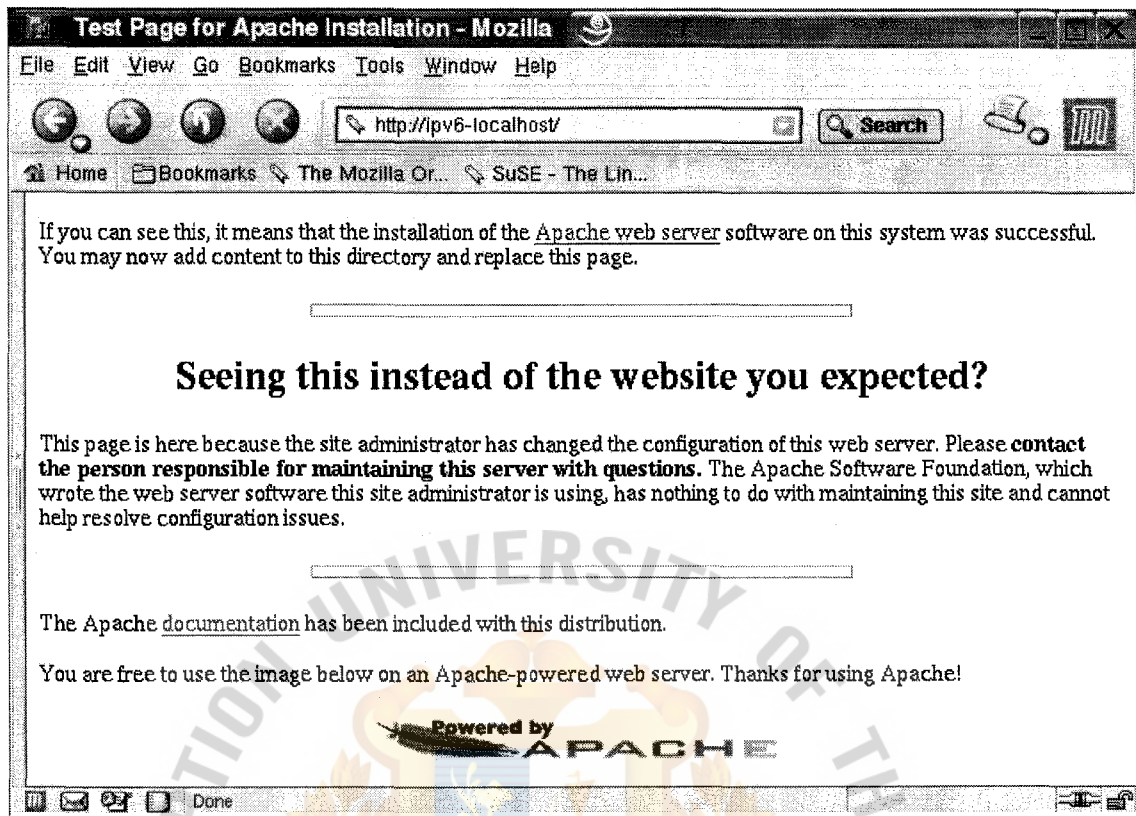


Figure 4.7. Apache Test Page for the Address ipv6-localhost.

Using the actual loopback address required some research as the browser is confused by the colons in the IPv6 address. In IPv4 addresses, the colon is used to separate the port number from the address, e.g. 192.168.0.1:80. The solution is simply to surround the IPv6 address in square brackets, e.g. [::1] for the loopback address. This technique is shown in Figure 4.8.

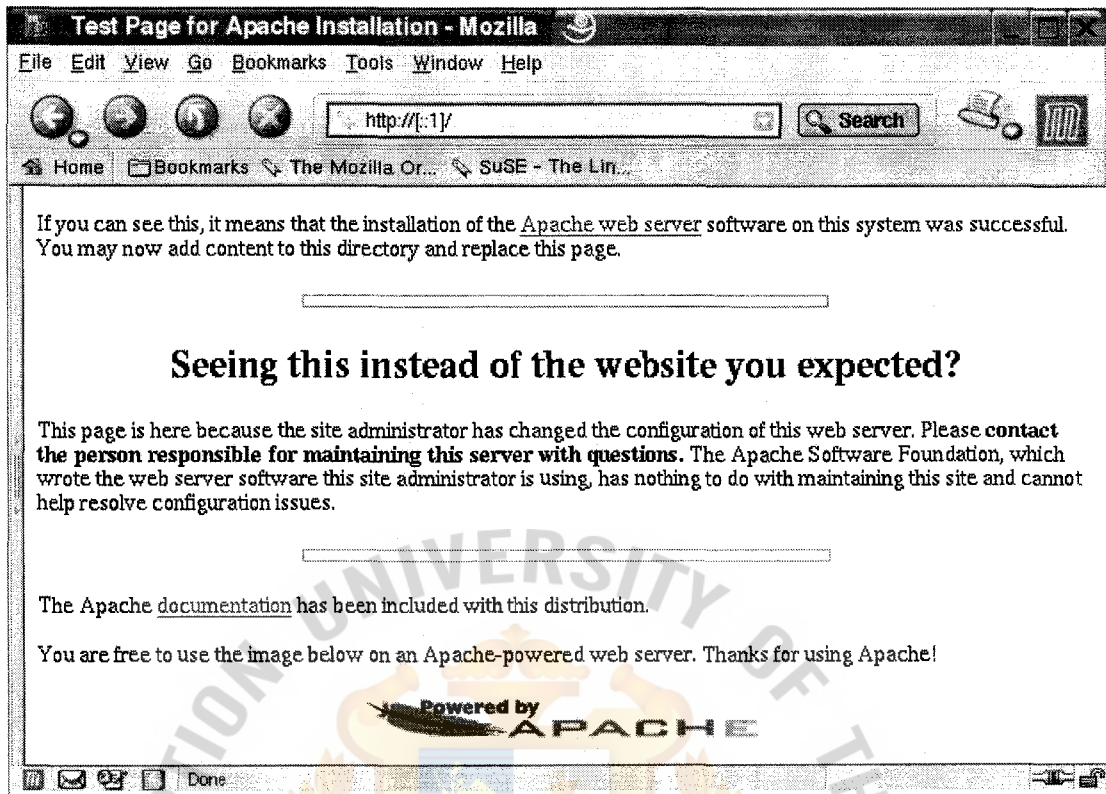


Figure 4.8. Apache Test Page for the Address [::1].

These tests included following the documentation hyperlinks. Everything functioned normatively, so installation of the web server was considered successful. The next step was to try to serve pages over the Ethernet network. The result is shown in Figure 4.9.



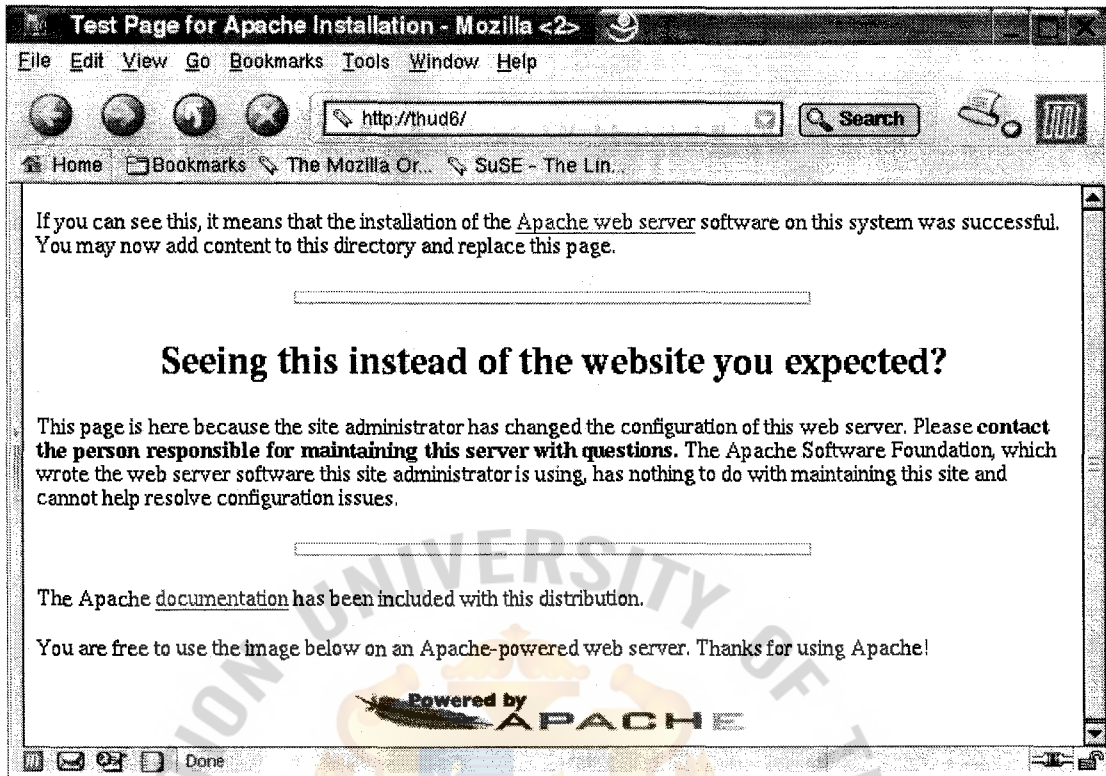


Figure 4.9. IPv6 Web Page over an Ethernet Network.

This demonstrates a successful service of web pages over an Ethernet LAN using the IPv6 enabled web server and the Mozilla 1.2.1 browser. It should be mentioned that there were problems with configuration, and the web service was only sporadically successful. Further investigation will be required to resolve the issue of dependable service.

The first portion of the project requirements has now been accomplished. The next section will discuss the testing of the system over the IPv6 6Bone network.

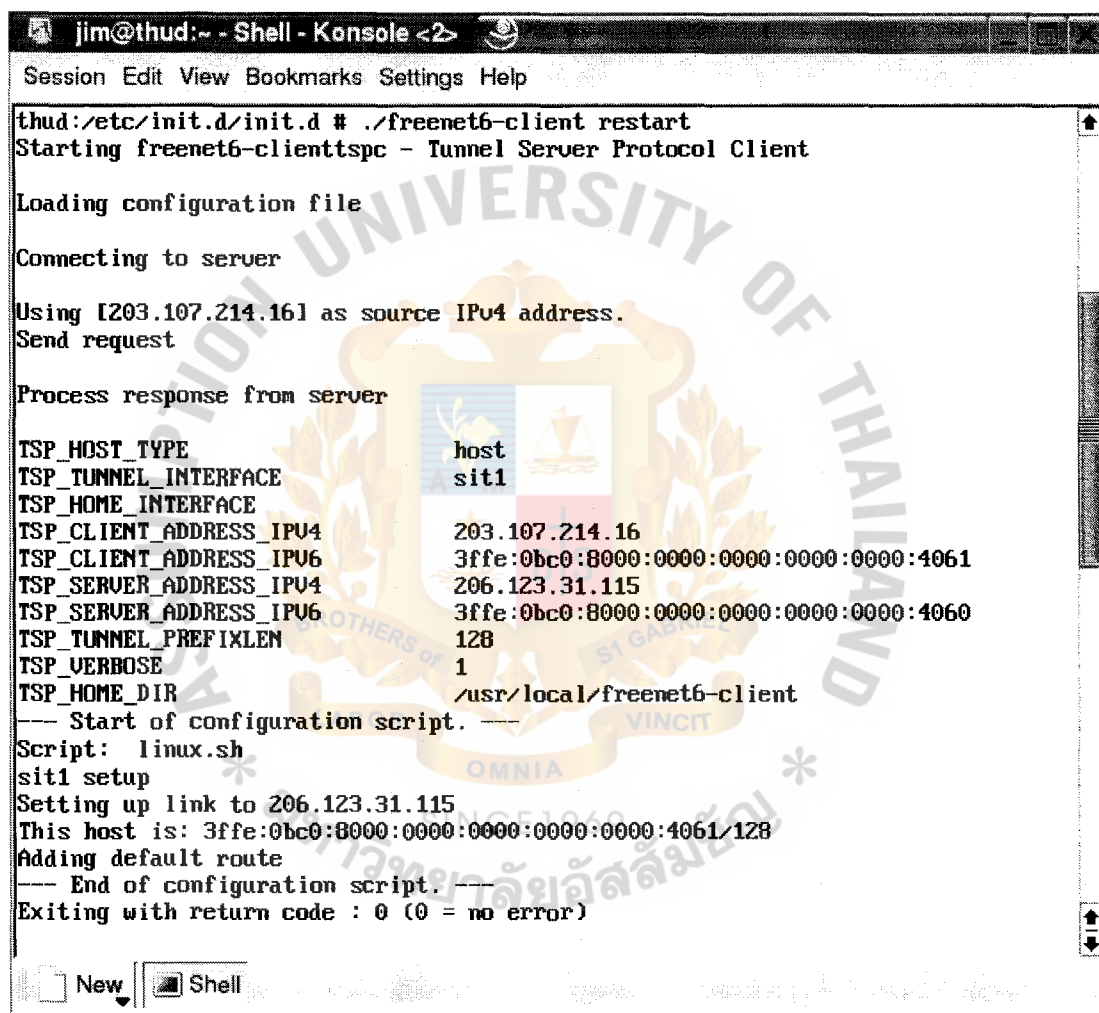
#### 4.6.2 Testing the System on the 6Bone IPv6 Backbone

There was a problem establishing a connection to the Freenet6 server over the ABAC dial up connection. These tunnels cannot pass through a NAT server or through a firewall that blocks tunneling. It is the hypothesis that something like this is the cause



of this failure. The KSC connection was successful, however. An additional account was opened with Loxley Information Services in order to handle the other side of the connection.

The initial connection results are displayed in Figure 4.10.



```
jim@thud:~ - Shell - Konsole <2>
Session Edit View Bookmarks Settings Help

thud:/etc/init.d/init.d # ./freenet6-client restart
Starting freenet6-clienttspc - Tunnel Server Protocol Client

Loading configuration file

Connecting to server

Using [203.107.214.16] as source IPv4 address.
Send request

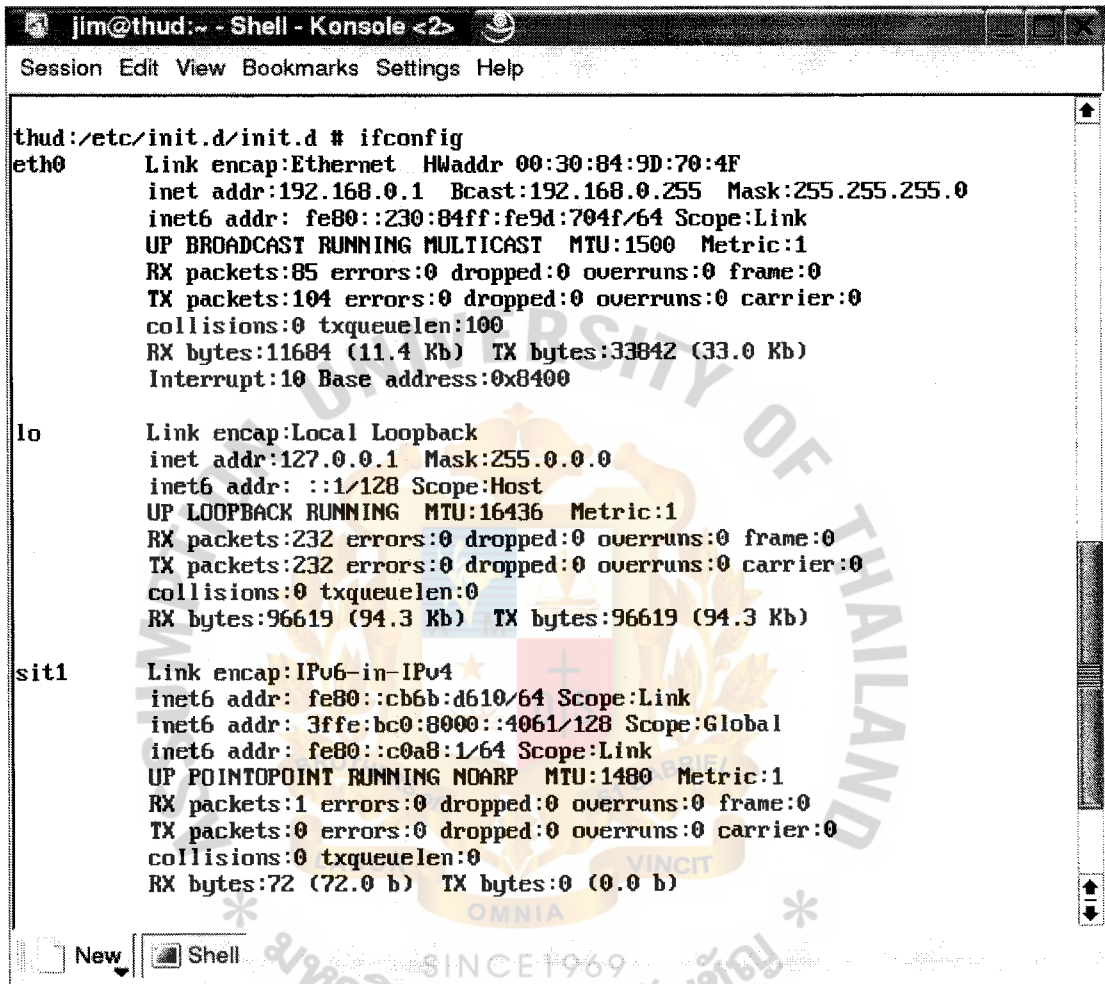
Process response from server

TSP_HOST_TYPE          host
TSP_TUNNEL_INTERFACE   sit1
TSP_HOME_INTERFACE
TSP_CLIENT_ADDRESS_IPv4 203.107.214.16
TSP_CLIENT_ADDRESS_IPv6 3ffe:0bc0:8000:0000:0000:0000:0000:4061
TSP_SERVER_ADDRESS_IPv4 206.123.31.115
TSP_SERVER_ADDRESS_IPv6 3ffe:0bc0:8000:0000:0000:0000:0000:4060
TSP_TUNNEL_PREFIXLEN   128
TSP_VERBOSE            1
TSP_HOME_DIR           /usr/local/freenet6-client
--- Start of configuration script. ---
Script: linux.sh
sit1 setup
Setting up link to 206.123.31.115
This host is: 3ffe:0bc0:8000:0000:0000:0000:0000:4061/128
Adding default route
--- End of configuration script. ---
Exiting with return code : 0 (0 = no error)
```

Figure 4.10. Initial Freenet6 Connection.

This connection also set up a new interface on the server host. The interface is called sit1, which displays the network reachable IPv6 Internet address that has been assigned to the server. The name of this interface refers to the Simple Internet

Transition Protocol that was discussed in section 3.1. This result is illustrated in the ifconfig dump shown in Figure 4.11.



```

thud:/etc/init.d/init.d # ifconfig
eth0      Link encap:Ethernet  HWaddr 00:30:84:9D:70:4F
          inet addr:192.168.0.1  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::230:84ff:fe9d:704f/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:85 errors:0 dropped:0 overruns:0 frame:0
          TX packets:104 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:11684 (11.4 Kb)  TX bytes:33842 (33.0 Kb)
          Interrupt:10 Base address:0x8400

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:232 errors:0 dropped:0 overruns:0 frame:0
          TX packets:232 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:96619 (94.3 Kb)  TX bytes:96619 (94.3 Kb)

sit1      Link encap:IPv6-in-IPv4
          inet6 addr: fe80::cb6b:d610/64 Scope:Link
          inet6 addr: 3ffe:bc0:8000::4061/128 Scope:Global
          inet6 addr: fe80::c0a8:1/64 Scope:Link
          UP POINTOPOINT RUNNING NOARP  MTU:1480  Metric:1
          RX packets:1 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:72 (72.0 b)  TX bytes:0 (0.0 b)

```

Figure 4.11. The sit1 Interface Added by Freenet6.

After this initial configuration, a test run was conducted in which the Mozilla browser attempted to retrieve the Apache test page over the circuit provided by the Freenet6 service. First, a connection was made to the Internet via the KSC account. Next the command “./etc/init.d/init.d/freenet6-client restart” was executed with root privileges. The result of this operation is shown in Figure 4.12.

```
jim@thud:~ - Shell - Konsole
Session Edit View Bookmarks Settings Help

Starting freenet6-clienttspc - Tunnel Server Protocol Client

Loading configuration file

Connecting to server

Using [203.107.215.199] as source IPv4 address.
Send request

Process response from server

TSP_HOST_TYPE          host
TSP_TUNNEL_INTERFACE   sit1
TSP_HOME_INTERFACE
TSP_CLIENT_ADDRESS_IPV4 203.107.215.199
TSP_CLIENT_ADDRESS_IPV6 3ffe:0bc0:8000:0000:0000:0000:0000:4061
TSP_SERVER_ADDRESS_IPV4 206.123.31.115
TSP_SERVER_ADDRESS_IPV6 3ffe:0bc0:8000:0000:0000:0000:0000:4060
TSP_TUNNEL_PREFIXLEN    128
TSP_VERBOSE             1
TSP_HOME_DIR            /usr/local/freenet6-client
--- Start of configuration script. ---
Script: linux.sh
sit1 setup
Setting up link to 206.123.31.115
Removing old IPv6 address 3ffe:bc0:8000::4061/128
This host is: 3ffe:0bc0:8000:0000:0000:0000:0000:4061/128
Adding default route
--- End of configuration script. ---
Exiting with return code : 0 (0 = no error)
```

Figure 4.12. Result of Freenet6 Configuration during Test Run.

It should be noted that although the client IPv4 address has changed since the initial configuration run (see Figure 4.10), the client IPv6 address remains static at 3FFE: BC0:8000::4061. Thus, the problem of accessing the 6Bone with dynamic IPv4 addresses has been overcome very elegantly.

Next, an effort was made to test the browser by requesting an IPv6 test page over the Freenet6 tunnel. Such a page is available from the KAME project. This page features a graphic of a turtle that is static for IPv4 connections but is animated if the

connection is made via IPv6. The connection was confirmed to be an IPv6 connection, as is illustrated by the two frames of the animation shown in Figures 4.13 and 4.14.

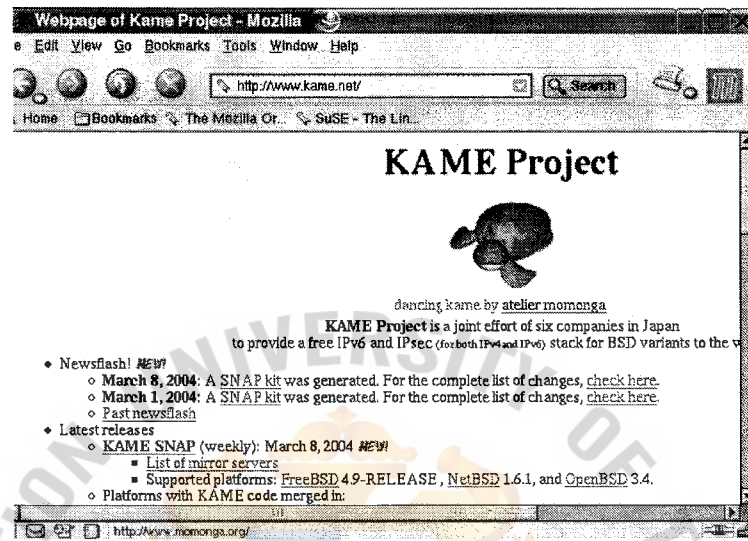


Figure 4.13. First Frame of the KAME Animation.

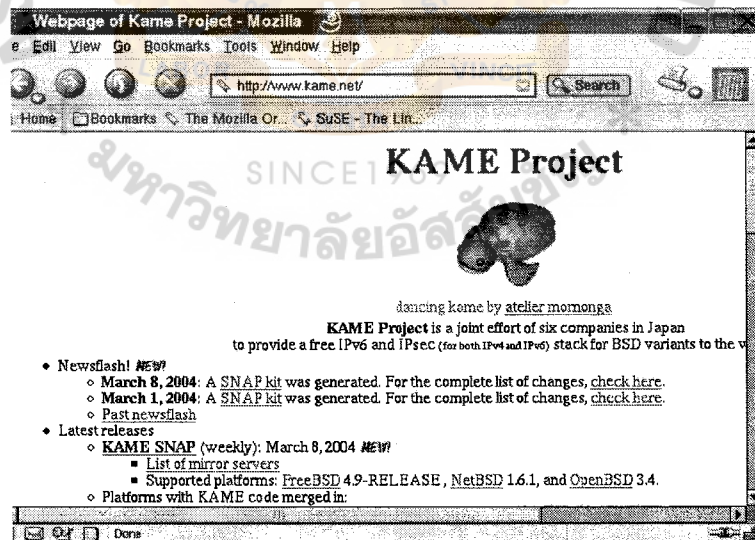


Figure 4.14. Second Frame of the KAME Animation



This outcome was confirmed by using the Traceroute 6 utility to trace the IPv6 portion of the path over the 6Bone. The result of that test is shown in Figure 4.15.

```

jim@thud:/etc/init.d/init.d - Shell - Konsole
Session Edit View Bookmarks Settings Help

thud:/etc/init.d/init.d # traceroute6 www.kame.net
traceroute to www.kame.net (2001:200:0:8002:203:47ff:fea5:3085), 30 hops max, 40
byte packets
 1 3ffe:bc0:8000::4060 567.017 ms 568.187 ms 579.725 ms
 2 Viagenie-gw.int.ipv6.ascc.net (2001:288:3b0::55) 579.798 ms 589.824 ms
599.827 ms
 3 3ffe:b00:c18::6b 629.833 ms 633.886 ms 639.878 ms
 4 3ffe:80a::e 659.874 ms 669.867 ms 669.663 ms
 5 2001:2a0:0:bb0a::1 749.217 ms 759.709 ms 769.847 ms
 6 2001:2a0:0:bb04::6 774.244 ms 779.730 ms 789.326 ms
 7 hitachi1.otemachi.wide.ad.jp (2001:200:0:1800::9c4:2) 739.257 ms 749.811
ms 749.715 ms
 8 pc3.yagami.wide.ad.jp (2001:200:0:1c04::1000:2000) 759.750 ms 769.812 ms
779.826 ms
 9 gr2000.k2c.wide.ad.jp (2001:200:0:4819::2000:1) 789.718 ms 789.815 ms 7
99.713 ms
10 orange.kame.net (2001:200:0:8002:203:47ff:fea5:3085) 809.822 ms 819.630 m
s 829.810 ms
thud:/etc/init.d/init.d #

```

Figure 4.15. Traceroute over IPv6.

The next step was to test the server's global IPv6 address from the client over an independent link. The server was started and the connection to KFC was initiated. When the connection was established, the Freenet6 tunnel was set up as above. The same series of actions were performed on the client from a neighboring location. The server's global IPv6 address (3FFE:BC0:8000::4061) was typed into the browser's Universal Resource Location (URL) window, causing an HTTP request to be sent over the link to the Apache server. The result of this operation is shown in Figure 4.16.



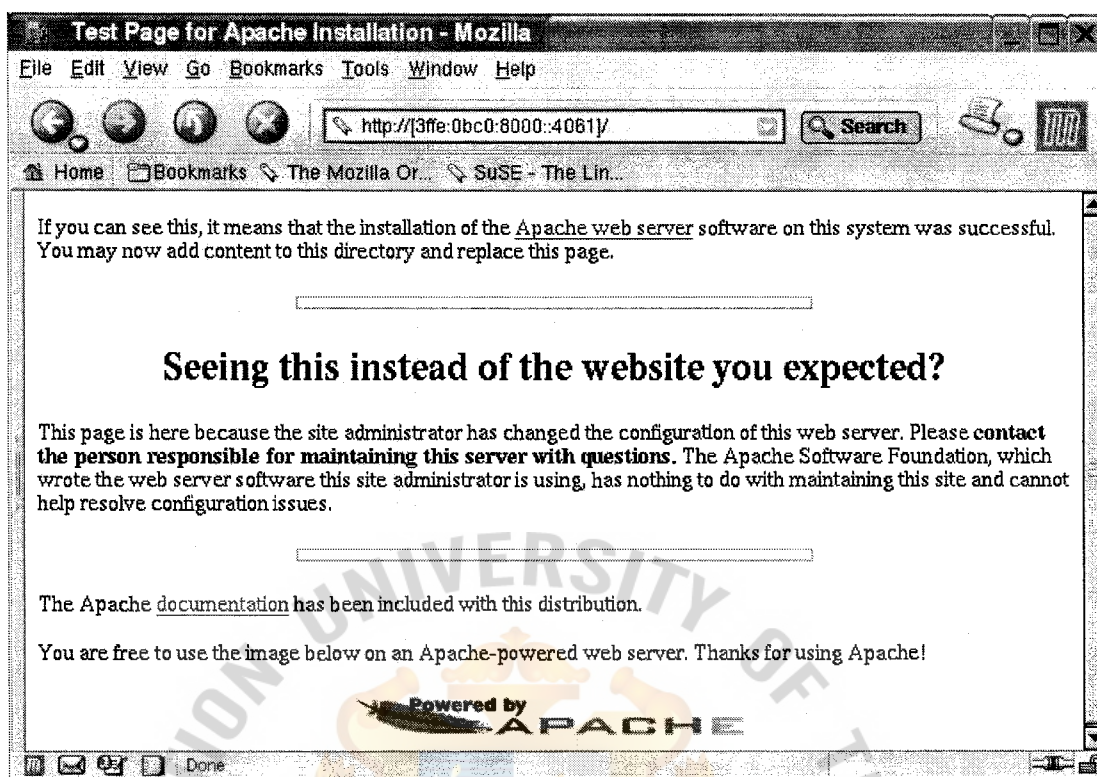


Figure 4.16. IPv6 Web Page Served to a Global IPv6 Address.

The successful outcome of this test fulfills all the remaining requirements for the project. The Apache server has successfully provided IPv6 HTTP service over both an Ethernet local area network and the Freenet6 tunnel server to the 6Bone. Additionally, the browser successfully traversed the 6Bone and retrieved an IPv6 only test page.

With the system thus configured, it is ready for further development as is recommended in section 5.2

## IV. CONCLUSIONS AND RECOMMENDATIONS

### 5.1 Conclusions

It is absolutely certain that the next generation IPv6 protocol will eventually overtake and supplant the present IPv6 protocol. Not only does it solve the critical problems of address space exhaustion and router table collapse that threaten the current Internet, but it provides well-designed and even elegant solutions to the emerging requirements of the future Internet.

The question of how soon this transition will occur is somewhat unclear. It is apparent that the early stages of a production IPv6 Internet are dawning. Most of the coding, testing and technical implementation has already been accomplished. However, the actual transition is still at an experimental stage. The software is not really mature; and we may expect that there will be considerable resistance to such a major change. There is an old adage that says, “if it isn’t broken, don’t fix it,” which sums up the attitude that we are bound to encounter.

In answer to that objection, we might reply that if we don’t fix it soon, it most certainly will be broken. Despite such clever stopgap measures as CIDR, NAT and DHCP, backbone routing tables are seriously strained and the Internet is still growing exponentially. Furthermore, the vision of a more secure Internet that will allow for the massive streaming of vast amounts of multimedia and real time applications data requires that the transition proceed apace.

One of the aims of this project was to tackle the somewhat daunting problem of designing and assembling an IPv6 capable web server system over the experimental 6Bone without the benefit of a static IPv4 address. To the extent that this was successful, it is very gratifying. The amount of frustration encountered, however,

indicates that IPv6 implementation is still very much at the beta stage. As commercial production IPv6 networks come on line, the task of acquiring IPv6 connectivity will become much simpler, but there will certainly be a need for personnel who are familiar with the details of such networks.

## 5.2 Recommendations

When should an organization make the move to IPv6? This is extremely difficult to answer. If an organization has enough IPv4 address space and no particular need for IP integrated security or multicast streams of data, the answer is not very soon. For such organizations, the gradual transition strategy can be very gentle indeed, allowing them to make the change one LAN at a time. There is no onus to not changing until the equipment becomes obsolete. The transition can proceed one machine at a time.

If however, an organization is unable to obtain sufficient globally reachable IPv4 addresses or has a need for state-of-the-art real time or multimedia data flows, the answer is likely to be to make the transition as soon as a commercial production IPv6 provider becomes available. Application servers are likely to be among the systems that are early adopters.\*

This strategic flexibility during the transition period allows solutions to be customized according to each organization's individual needs. The only certainty is that one day, perhaps fairly far into the future, IPv4 will be deployed no more.

There are many possible topics of major interest for further study. The writer intends to further explore setting up a router on the system constructed for this project, along with the implementation of security and autoconfiguration. On a more general note, the drive for integration of distributed systems such as the .Net and Java initiatives immediately raises the question of the degree to which they are IPv6 compliant. There

is also a great deal of work to be done in benchmarking and tuning the performance of IPv6 compliant software.

Designing networks that take advantage of the unique capabilities of IPv6 Internetworking is one of the future's most interesting challenges.





## APPENDIX A

### LOCATING THE SOFTWARE PACKAGES



Table A.1. URL Addresses for Software Used in the Project

Software Title	URL where available
Apache Web Server	<a href="http://www.apache.org">www.apache.org</a>
Freenet6 Utility	<a href="http://freenet6.net">freenet6.net</a>
Linux Kernel	<a href="http://V6rpm.jindai.net/v6rpm.html">V6rpm.jindai.net/v6rpm.html</a>
Mozilla	<a href="http://www.mozilla.org">www.mozilla.org</a>
Iputils	<a href="ftp.inr.ac.ru/ip-routing">ftp.inr.ac.ru/ip-routing</a>
NetKit	<a href="http://freshmeat.net/projects/netkit">freshmeat.net/projects/netkit</a>
Net-tools	<a href="http://www.tazenda.demon.co.uk/phil/net-tools">www.tazenda.demon.co.uk/phil/net-tools</a>
Opera Browser	<a href="http://www.opera.com">www.opera.com</a>
SuSE Linux	<a href="http://www.suse.com">www.suse.com</a>
USAGI IPv6 stack	<a href="http://www.linux.ipv6.org">www.linux.ipv6.org</a>

Table A.2 URL Addresses for Related Organizations

Organization	Contact URL
6Bone	<a href="http://www.6bone.net">www.6bone.net</a>
IETF (RFCs)	<a href="http://www.ietf.org">www.ietf.org</a>
IPv6.org	<a href="http://www.ipv6.org">www.ipv6.org</a>
IPv6 Status Summary	<a href="http://www.ipv6forum.com">www.ipv6forum.com</a>
Kame Project	<a href="http://www.kame.net">www.kame.net</a>
TAHI Project	<a href="http://www.tahi.org">www.tahi.org</a>
Viagenie	<a href="http://www.viagenie.qc.ca">www.viagenie.qc.ca</a>
Wide	<a href="http://www.v6.wide.ad.jp">www.v6.wide.ad.jp</a>



## APPENDIX B

### IP PACKET HEADER DUMPS

## B.1 IPv4 Ping Headers.

The following are the IPv4 headers from a part of a ping session. They illustrate arp and echo requests and replies. They were obtained by the command “tcpdump -x > filename”, where the -x signals an octal dump and > indicates that the results are to be redirected into a file called filename.

```
21:00:03.087330 arp who-has thud.davidson tell 192.168.0.99
0001 0800 0604 0001 0010 60f3 0fad c0a8
0063 0000 0000 0000 c0a8 0001 0000 0000
0230 84ff fe9d 704f 8800 64ac 4000
21:00:03.087371 arp reply thud.davidson is-at 0:30:84:9d:70:4f
0001 0800 0604 0002 0030 849d 704f c0a8
0001 0010 60f3 0fad c0a8 0063
21:00:03.087804 192.168.0.99 > thud.davidson: icmp: echo request
4500 0054 0012 0000 4001 f8e2 c0a8 0063
c0a8 0001 0800 3131 dd03 0000 3ecf e647
0006 d9aa 0809 0a0b 0c0d 0e0f 1011 1213
1415 1617 1819 1a1b 1c1d 1e1f 2021 2223
2425 2627 2829 2a2b 2c2d 2e2f 3031 3233
3435
21:00:03.087846 thud.davidson > 192.168.0.99: icmp: echo reply
4500 0054 0951 0000 4001 efa3 c0a8 0001
c0a8 0063 0000 3931 dd03 0000 3ecf e647
0006 d9aa 0809 0a0b 0c0d 0e0f 1011 1213
1415 1617 1819 1a1b 1c1d 1e1f 2021 2223
2425 2627 2829 2a2b 2c2d 2e2f 3031 3233
3435
21:00:04.093844 192.168.0.99 > thud.davidson: icmp: echo request
4500 0054 0013 0000 4001 f8e1 c0a8 0063
c0a8 0001 0800 1806 dd03 0001 3ecf e648
0006 f2d3 0809 0a0b 0c0d 0e0f 1011 1213
1415 1617 1819 1a1b 1c1d 1e1f 2021 2223
2425 2627 2829 2a2b 2c2d 2e2f 3031 3233
3435
21:00:04.093893 thud.davidson > 192.168.0.99: icmp: echo reply
4500 0054 0952 0000 4001 efa2 c0a8 0001
c0a8 0063 0000 2006 dd03 0001 3ecf e648
0006 f2d3 0809 0a0b 0c0d 0e0f 1011 1213
1415 1617 1819 1a1b 1c1d 1e1f 2021 2223
2425 2627 2829 2a2b 2c2d 2e2f 3031 3233
3435
```

## B.2 IPv6 Ping Headers

The following are the IPv6 headers from a part of a ping session. They illustrate IPv6 neighbor discovery, with its three-way handshake, along with echo requests and replies.

It is of interest to compare these with the ones on the preceding page.

```
18:30:15.151202 fe80::10:60f3:fad > ff02::1:ff9d:704f: icmp6: neighbor
sol: who has fe80::230:84ff:fe9d:704f
    6000 0000 0020 3aff fe80 0000 0000 0000
    0000 0010 60f3 0fad ff02 0000 0000 0000
    0000 0001 ff9d 704f 8700 3433 0000 0000
    fe80 0000 0000 0000 0230 84ff fe9d 704f
    0101 0010 60f3 0fad
18:30:15.151292 fe80::230:84ff:fe9d:704f > fe80::10:60f3:fad: icmp6:
neighbor adv: tgt is fe80::230:84ff:fe9d:704f
    6000 0000 0020 3aff fe80 0000 0000 0000
    0230 84ff fe9d 704f fe80 0000 0000 0000
    0000 0010 60f3 0fad 8800 c819 6000 0000
    fe80 0000 0000 0000 0230 84ff fe9d 704f
    0201 0030 849d 704f
18:30:15.151332 fe80::10:60f3:fad > ff02::1:fe9d:704f: icmp6: neighbor
sol: who has fe80::230:84ff:fe9d:704f
    6000 0000 0020 3aff fe80 0000 0000 0000
    0000 0010 60f3 0fad ff02 0000 0000 0000
    0000 0001 fe9d 704f 8700 3533 0000 0000
    fe80 0000 0000 0000 0230 84ff fe9d 704f
    0101 0010 60f3 0fad
18:30:15.151851 fe80::10:60f3:fad > fe80::230:84ff:fe9d:704f: icmp6:
echo request
    6000 0000 0040 3a40 fe80 0000 0000 0000
    0000 0010 60f3 0fad fe80 0000 0000 0000
    0230 84ff fe9d 704f 8000 6555 df06 0000
    2b66 d23e e4b1 0a00 0809 0a0b 0c0d 0e0f
    1011 1213 1415 1617 1819 1a1b 1c1d 1e1f
    2021
18:30:15.151887 fe80::230:84ff:fe9d:704f > fe80::10:60f3:fad: icmp6:
echo reply
    6000 0000 0040 3a40 fe80 0000 0000 0000
    0230 84ff fe9d 704f fe80 0000 0000 0000
    0000 0010 60f3 0fad 8100 6455 df06 0000
    2b66 d23e e4b1 0a00 0809 0a0b 0c0d 0e0f
    1011 1213 1415 1617 1819 1a1b 1c1d 1e1f
    2021
    2021
```

## BIBLIOGRAPHY

1. Bieringer, Peter, The IPv6 Linux HOWTO. Available at: <http://www.bieringer.de/linux/IPv6>, Updated January 2004.
2. Bieringer, Peter. "IPv6: The Future of Internet Addresses." Linux Magazine, (May, 2002): 32-36ff.
3. Black, Ulysses. TCP/IP and Related Protocols, 3rd Edition. NY: McGraw-Hill, 1998.
4. Comer, Douglas E. Internetworking with TCP/IP: Principles, Protocols and Architectures, 4<sup>th</sup> Edition. Upper Saddle River, NJ: Prentice Hall, 2000.
5. Conta, A. and S. Deering. RFC 1885: Internet Control Message Protocol (ICMPv6). December 1995.
6. Delgrossi L. and L. Berger. RFC 1819: Internet Stream Protocol Version 2 (ST2) Protocol Specification - Version ST2+, August 1995.
7. Deering S. and R. Hinden. RFC 1883: Internet Protocol, Version 6 (IPv6) Specification, December 1995.
8. Eltz, R. RFC 1924: A Compact Representation of IPv6 Addresses, April 1996.
9. Feit, Sidnie. TCP/IP: Architecture, Protocols, and Implementation with IPv6 and IP Security, 2nd Edition. NY: McGraw-Hill, 1997.
10. Forouzan, Behrouz A. Data Communication and Networking, 2nd Edition. Singapore: McGraw-Hill, 2001.
11. Forouzan, Behrouz A. TCP/IP Protocol Suite, 2nd Edition. Singapore: McGraw-Hill, 2003.
12. Gai, Silvano. Internetworking IPv6 with Cisco Routers. NY: McGraw-Hill, 1998.
13. Gilligan R. and E. Nordmark. RFC 2893: Transition Mechanisms for IPv6 Hosts and Routers, August 2000.
14. Gouda, Mohamed G. Elements of Network Protocol Design. NY: John Wiley and Sons, 1998.
15. Haddad, Ibrahim. "Linux IPv6: Which One to Deploy?" Linux Journal, no. 96 (April, 2002): 86-90.



16. Haddad, Ibrahim and Marc Blanchet. "Supporting IPv6 on a Linux Server Node." Linux Journal, no. 100 (August, 2002): 100-107.
17. Haddad, Ibrahim and David Gordon. "Apache Talking IPv6" Linux Journal, no. 105 (January, 2003): 86-90.
18. Hinden R. and J. Postal. RFC 1897: IPv6 Testing Address allocation, January 1996.
19. Hinden R. and S. Deering. RFC 2373: IP Version 6 Addressing Architecture, July 1998.
20. Hinden R. and S. Deering. RFC 3513: Internet Protocol Version 6 (IPv6) Addressing Architecture, April 2003.
21. Huitema C. RFC 1715: The H Ratio for Address Assignment Efficiency, November 1994.
22. Huitema, C. IPv6: The New Internet Protocol. Upper Saddle River, NJ: Prentice Hall, 1996.
23. Mogul J. and S. Deering. RFC 1191: Path MTU discovery, November 1990.
24. Thomas, S. A. IPng and the TCP/IP Protocols. NY: John Wiley & Sons, 1996.
25. Thompson S. and C. Huitema. RFC 1886: DNS Extensions to support IP version 6, September 1995.
26. Todd, Peter. "Get IPv6 Now with Freenet6" Linux Journal, no. 105 (January, 2003): 65-67.