Tunneling:
A Transition Mechanism to Deploy IPv6 Networks

Authors

Ioan Raicu

Ahmad Naveed

Course: CSC8260

Advisor

Dr. Sherali Zeadally

Wayne State University

Document Created: 11/5/2001

Last Date Modified: 2/20/2002

1.0 Introduction

Although the Internet Protocol known as IPv4 served its purpose for over 20 years, its days are numbered. With IPv6 reaching a mature enough level, there is a need to evaluate the performance: benefits or drawbacks that the new IPv6 protocol will have in comparison to the well established IPv4 protocol. Theoretically, the overhead between the two different protocols should be directly proportional to the difference in the packet's header size, however according to our findings, the empirical performance difference between IPv4 and IPv6 is much larger than anticipated. Our experiments were conducted using an Ericsson dual stack (IPv4/IPv6) router with two end nodes running both Windows 2000. Our goal was to perform an unbiased empirical performance evaluation between the two protocol stacks (IPv4 and IPv6) on identical hardware and under identical settings. Furthermore, we chose to pursue transition mechanisms in order to upgrade the existing IPv4 infrastructure to the next generation Internet Protocol, IPv6.

IPv4 was designed at a time when there were very few nodes throughout the network. Typical network conditions were low bandwidth, high delay, and large bit error rates (BER). Most common applications at the time were FTP, e-mail, etc… all elastic applications.

In the early 1990's, the computer industry expanded exponentially with the personal computer (PC). The internet also gained momentum as businesses saw potential in conducting business electronically, known as E-Commerce. The market demand was the biggest factor in the Internet's revolution. As the Internet's explosion was becoming evident in the early 1990's, it was constituted that the IPv4 address space would be, depleted by the end of the millennium. Mechanisms such as Network Address Translator (NAT) have elongated the life of IPv4, but its days are numbered.

Today, the market looks completely different than it did in the 1980's. Although FTP and e-mail are still very popular today, new inelastic applications such as video conferencing, video-on-demand, Voice-over-IP, e-Commerce, and support for magnitudes more users than ever anticipated, have led the Internet Engineering Task Force (IETF) to seek a new Internet Protocol, namely IPv6.

As IPv6 is finally beginning to mature, it is evident that methods of upgrading the Internet need to be found. One idea would be to turn off the entire Internet at 12AM, upgrade the network infrastructure (routers, protocol stacks, etc…), and turn the Internet back on at 6AM and hope everything works. This is unrealistic due to the fact that it would cost more money than it is imaginable, the time would be way too short, and nothing ever works as good as it is in theory. More gradual transition methods have evolved, ones which are likely to happen over the course of 10 years or so. Among them some of the transition mechanisms are:

- Dual Stacks
- DTI & Bump-in-dual-stack
- NAT – Protocol Translator
- SIIT – Stateless IP/ ICMP Translator
- AIIH – Assignment of IPv4 Global Addresses to IPv6 Hosts
- Tunnel Broker
- 6-to-4 Mechanism
- IPv6 in IPv4 tunneling

We chose to pursue the IPv6 in IPv4 tunneling as a transition mechanism because it would be the most cost effective and can implement islands of IPv6 networks that can be connected over the existing ocean of IPv4 networks, the existing infrastructure. With time, as the islands grow, the ocean will

diminish to a point that all the islands will touch, at which point it is evident that native IPv6 networks will finally reign and benefit from its new features 100%.

2.0 Overview

In this section, we will discuss the basics of IPv6 and how it compares with IPv4 in order to better understand and appreciate the results obtained. The details about IPv6 are in general the way they were proposed in the RFCs by IETF, however we chose to use Microsoft Windows 2000 as the platform to implement our tests. Due to their early stages of development, the IPv6 protocol stack in Windows 2000 still has many problems, such as fragmentation issues, no support for IPSec, a native security feature, etc…

Microsoft has two different implementations of an IPv6 stack both for Windows NT 4.0 and Windows 2000. The older stack, known as the "Microsoft Research IPv6 Release 1.4", works under both NT 4.0 and Win2K; the newer stack, known as the "Microsoft IPv6 Technology Preview for Windows 2000" works under Windows 2000. Both stacks require an existing IPv4 stack to be previously installed. Once installed, besides giving the Windows environment the support for IPv6, it creates a whole new set of routines, such as "ping6", "tracert6", which are similar in function to "ping" and "tracert", but work with the new IPv6 stack. The good part about the IPv6 implementation that Microsoft created is that they embedded the IPv6 socket creation in the Winsock2 API. That means that they added a few more functions when you create the sockets, however, the fundamentals remained the same, and thus a programmer that can make an IPv4 application can most likely learn how to make a simple IPv6 application as well.

2.1 Transport Protocols

2.1.1 TCP Protocol

TCP is a connection oriented protocol that guarantees sent data to be correct, arrive in order, and provides flow control. A connection needs to be established, and once the connection is no longer needed, it must be deleted. The size of the TCP header is 20 bytes.

2.1.2 UDP Protocol

UDP is a connectionless service that sends datagrams across a network. Each datagram is independent of each other, which means that some could get lost, arrive out of order, etc… There are no guarantees with UDP. The header is only 8 bytes compared to the TCP header, thus it is a lighter weight transport protocol.

2.2 Network Protocols

2.2.1 IPv4 – Internet Protocol version 4 [13]

Internet Protocol version 4 is the current version of IP, which was finally revised in 1981. It has a 32 bit address looking like 255.255.255.255, and it supports up to 4,294,967,296 addresses.

The IPv6 header is a streamlined version of the IPv4 header. It eliminates fields that are unneeded or rarely used and adds fields that provide better support for real-time traffic. An overview of the IPv4 header is helpful in understanding the IPv6 header.

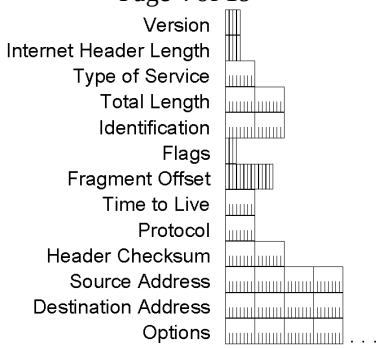

Figure 1 The IPv4 header

The fields in the IPv4 header are:

- **Version** – Indicates the version of IP and is set to 4. The size of this field is 4 bits.
- **Internet Header Length** – Indicates the number of 4-byte blocks in the IP header. The size of this field is 4 bits. Because an IP header is a minimum of 20 bytes in size, the smallest value of the Internet Header Length (IHL) field is 5. IP options can extend the minimum IP header size in increments of 4 bytes. If an IP option does not use all 4 bytes of the IP option field, the remaining bytes are padded with 0's, making the entire IP header an integral number of 32-bits (4 bytes). With a maximum value of 0xF, the maximum size of the IP header including options is 60 bytes (15*4).
- **Type of Service** – Indicates the desired service expected by this packet for delivery through routers across the IP internetwork. The size of this field is 8 bits, which contain bits for precedence, delay, throughput, and reliability characteristics.
- **Total Length** – Indicates the total length of the IP packet (IP header + IP payload) and does not include link layer framing. The size of this field is 16 bits, which can indicate an IP packet that is up to 65,535 bytes long.
- **Identification** – Identifies this specific IP packet. The size of this field is 16 bits. The Identification field is selected by the originating source of the IP packet. If the IP packet is fragmented, all of the fragments retain the Identification field value so that the destination node can group the fragments for reassembly.
- **Flags** – Identifies flags for the fragmentation process. The size of this field is 3 bits, however, only 2 bits are defined for current use. There are two flags—one to indicate whether the IP packet might be fragmented and another to indicate whether more fragments follow the current fragment.
- **Fragment Offset** – Indicates the position of the fragment relative to the original IP payload. The size of this field is 13 bits.
- **Time to Live** – Indicate the maximum number of links on which an IP packet can travel before being discarded. The size of this field is 8 bits. The Time-to-Live field (TTL) was originally used as a time count with which an IP router determined the length of time required (in seconds) to forward the IP packet, decrementing the TTL accordingly. Modern routers almost always forward an IP packet in less than a second and are required by RFC 791 to decrement the TTL by at least one. Therefore, the TTL becomes a maximum link count with the value set

by the sending node. When the TTL equals 0, the packet is discarded and an ICMP Time Expired message is sent to the source IP address.

- **Protocol** – Identifies the upper layer protocol. The size of this field is 8 bits. For example, TCP uses a Protocol of 6, UDP uses a Protocol of 17, and ICMP uses a Protocol of 1. The Protocol field is used to demultiplex an IP packet to the upper layer protocol.
- **Header Checksum** – Provides a checksum on the IP header only. The size of this field is 16 bits. The IP payload is not included in the checksum calculation as the IP payload and usually contains its own checksum. Each IP node that receives IP packets verifies the IP header checksum and silently discards the IP packet if checksum verification fails. When a router forwards an IP packet, it must decrement the TTL. Therefore, the Header Checksum is recomputed at each hop between source and destination.
- **Source Address** – Stores the IP address of the originating host. The size of this field is 32 bits.
- **Destination Address** – Stores the IP address of the destination host. The size of this field is 32 bits.
- **Options** – Stores one or more IP options. The size of this field is a multiple of 32 bits. If the IP option or options do not use all 32 bits, padding options must be added so that the IP header is an integral number of 4-byte blocks that can be indicated by the Internet Header Length field.

2.2.2 IPv6 – Internet Protocol version 6 [13]

Internet Protocol version 6 is designed as an evolutionary upgrade to the Internet Protocol (IPv4) and will, in fact, coexist with the older IPv4 for some time. IPv6 is designed to allow the Internet

to grow steadily, both in terms of the number of hosts connected and the total amount of data traffic transmitted; it will have a 128 bit address looking like FFFF:FFFF:FFFF:FFFF, and it will support up to 340,282,366,920938,463,463,374,607,431,768,211,456 unique addresses.

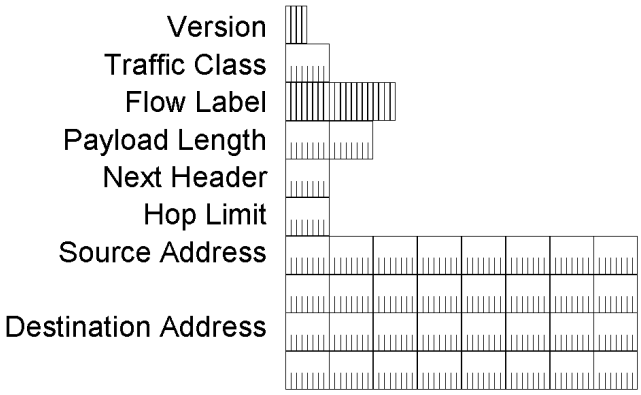

Figure 2 The IPv6 header

The IPv6 header is always present and is a fixed size of 40 bytes. The fields in the IPv6 header are described briefly below.

The fields in the IPv6 header are:

- **Version** – 4 bits are used to indicate the version of IP and is set to 6.
- **Traffic Class** – Indicates the class or priority of the IPv6 packet. The size of this field is 8 bits. The Traffic Class field provides similar functionality to the IPv4 Type of Service field. In RFC

2460, the values of the Traffic Class field are not defined. However, an IPv6 implementation is required to provide a means for an application layer protocol to specify the value of the Traffic Class field for experimentation.

? **Flow Label** – Indicates that this packet belongs to a specific sequence of packets between a source and destination, requiring special handling by intermediate IPv6 routers. The size of this field is 20 bits. The Flow Label is used for non-default quality of service connections, such as those needed by real-time data (voice and video). For default router handling, the Flow Label is set to 0. There can be multiple flows between a source and destination, as distinguished by separate non-zero Flow Labels.

? **Payload Length** – Indicates the length of the IP payload. The size of this field is 16 bits. The Payload Length field includes the extension headers and the upper layer PDU. With 16 bits, an IPv6 payload of up to 65,535 bytes can be indicated. For payload lengths greater than 65,535 bytes, the Payload Length field is set to 0 and the Jumbo Payload option is used in the Hop-by-Hop Options extension header.

? **Next Header** – Indicates either the first extension header (if present) or the protocol in the upper layer PDU (such as TCP, UDP, or ICMPv6). The size of this field is 8 bits. When indicating an upper layer protocol above the Internet layer, the same values used in the IPv4 Protocol field are used here.

  o **Extension Header** – Zero or more extension headers can be present and are of varying lengths. A Next Header field in the IPv6 header indicates the next extension header. Within each extension header is another Next Header field that indicates the next extension header. The last extension header indicates the upper layer protocol (such as TCP, UDP, or ICMPv6) contained within the upper layer protocol data unit. The IPv6 header and extension headers replace the existing IPv4 IP header with options. The new extension header format allows IPv6 to be augmented to support future needs and capabilities. Unlike options in the IPv4 header, IPv6 extension headers have no maximum size and can expand to accommodate all the extension data needed for IPv6 communication.

? **Hop Limit** – Indicates the maximum number of links over which the IPv6 packet can travel before being discarded. The size of this field is 8 bits. The Hop Limit is similar to the IPv4 TTL field except that there is no historical relation to the amount of time (in seconds) that the packet is queued at the router. When the Hop Limit equals 0, the packet is discarded and an ICMP Time Expired message is sent to the source address.

? **Source Address** –Stores the IPv6 address of the originating host. The size is 128 bits.

? **Destination Address** – Stores the IPv6 address of the current destination host. The size of this field is 128 bits. In most cases the Destination Address is set to the final destination address. However, if a Routing extension header is present, the Destination Address might be set to the next router interface in the source route list.

2.2.3 Comparison between IPv4 and IPv6 [13]

Table 1 shows the composition of the packet, which is carried across the network in Ethernet.

	IPv4 TCP	IPv6 TCP	IPv4 UDP	IPv6 UDP
TCP/UDP Payload	1460	1440	1472	1452
TCP/UDP Header	20	20	8	8
IP Payload	1480	1460	1480	1460
IP Header	20	40	20	40
Ethernet Header	14	14	14	14
Total Ethernet MTU	1514	1514	1514	1514
Overhead %	3.7%	5.14%	2.85%	4.27%

Table 1. Differences between IPv4 and IPv6 ethernet overhead[17]

Table 2 shows the highlights in the differences between IPv4 and IPv6. There are many other differences, but they are outside the scope of this document.

IPv4	**IPv6**
Source and destination addresses are 32 bits (4 bytes) in length.	Source and destination addresses are 128 bits (16 bytes) in length.
IPSec support is optional.	IPSec support is required.
No identification of payload for QoS handling by routers is present within the IPv4 header.	Payload identification for QoS handling by routers is included in the IPv6 header using Flow Label field.
Fragmentation is supported at both routers and the host.	Fragmentation is not supported at routers. It is only supported at the sending host.
Header includes a checksum. Must be computed at every intervening node on a per packet basis (very time consuming).	Header does not include a checksum.
Header includes options.	All optional data is moved to IPv6 extension headers.
Address Resolution Protocol (ARP) uses broadcast ARP Request frames to resolve an IPv4 address to the link layer.	ARP Request frames are replaced with multicast Neighbor Solicitation messages.
Internet Group Management Protocol (IGMP) is used to manage local subnet group membership.	IGMP is replaced with Multicast Listener Discovery (MLD) messages.
ICMP Router Discovery is used to determine the IPv4 address of the best default gateway and is optional.	ICMPv4 Router Discovery is replaced with ICMPv6 Router Solicitation and Router Advertisement.
Broadcast addresses are used to send traffic to all nodes on a subnet.	There are no IPv6 broadcast addresses. Instead, a link-local scope all-nodes multicast address is used.
Must be configured either manually or through DHCP.	Does not require manual configuration or DHCP.
Uses host address (A) resource records in the Domain Name System (DNS) to map host names to IPv4 addresses.	Uses host address (AAAA) resource records in the DNS to map host names to IPv6 addresses.
Uses pointer (PTR) resource records in the IN-ADDR.ARPA DNS domain to map IPv4 addresses to host names.	Uses pointer (PTR) resource records in the IP6.INT DNS domain to map IPv6 addresses to host names.

Table 2. Differences between IPv4 and IPv6 protocol[13]

Now that the main differences in the protocols are clear, table 3 will describe the differences between IPv4 and IPv6 header fields.

IPv4 Header Field	IPv6 Header Field
Version	Same field but with different version numbers.
Internet Header Length	Removed in IPv6. IPv6 does not include a Header Length field because the IPv6 header is always a fixed size of 40 bytes. Each extension header is either a fixed size or indicates its own size.
Type of Service	Replaced by the IPv6 Traffic Class field.
Total Length	Replaced by the IPv6 Payload Length field, which only indicates the size of the payload.
Identification	Removed in IPv6. Fragmentation information is not included in the IPv6 header. It is contained in a Fragment extension header.
Fragmentation Flags	Removed in IPv6. Fragmentation information is not included in the IPv6 header. It is contained in a Fragment extension header.
Fragment Offset	Removed in IPv6. Fragmentation information is not included in the IPv6 header. It is contained in a Fragment extension header.
Time to Live	Replaced by the IPv6 Hop Limit field.
Protocol	Replaced by the IPv6 Next Header field.
Header Checksum	Removed in IPv6. In IPv6, bit-level error detection for the entire IPv6 packet is performed by the link layer.
Source Address	The field is the same except that IPv6 addresses are 128 bits in length.
Destination Address	The field is the same except that IPv6 addresses are 128 bits in length.
Options	Removed in IPv6. IPv4 options are replaced by IPv6 extension headers.

Table 3. Differences between IPv4 and IPv6 headers [13]

2.3 Transition Mechanisms

As IPv6 is finally beginning to mature, it is evident that methods of upgrading the Internet need to be found. One idea would be to turn off the entire Internet at 12AM, upgrade the network infrastructure (routers, protocol stacks, etc…), and turn the Internet back on at 6AM and hope everything works. This is unrealistic due to the fact that it would cost more money than it is imaginable, the time would be way too short, and nothing ever works as good as it is in theory. More gradual transition methods have evolved, ones which are likely to happen over the course of 10 years or so. Some of the transition mechanisms are:

- Dual Stacks – easiest to implement, however complexity increases due to both infrastructures; (RFC1933)
- DTI & Bump-in-dual-stack
- SIIT – Stateless IP/ ICMP Translator
- AIIH – Assignment of IPv4 Global Addresses to IPv6 Hosts
- NAT – Protocol Translator – has scaling and DNS issues, and has single point of failure disadvantage; (RFC2766)

- Tunnel Broker – dynamically gain access to tunnel servers, but has authentication and scaling issues; (RFC3053)
- 6-to-4 Mechanism – dynamic stateless tunnels over IPv4 infrastructure to connect 6-to-4 domains; (RFC3056)
- IPv6 in IPv4 tunneling – Allows existing infrastructure to be utilized via manually configured tunnels (RFC2473, RFC2529)
    - Host-Host Tunneling
    - Router-Router Tunneling
    - Host-Router and vice versa Tunneling

2.3.1 IPv6 in IPv4 tunneling

IPv6 in IPv4 tunneling is one of the easiest transition mechanism by which two IPv6 hosts / networks can be connected with each other while running on existing IPv4 networks through establishing some special routes called tunnels. In this technique, IPv6 packets are encapsulated in IPv4 packets and then are sent over IPv4 networks like ordinary IPv4 packets through tunnels. At the end of tunnel these packets are decapsulated to the original IPv6 packets.

The following are some important characteristics of tunneling mechanism:

- When encapsulating a datagram, the TTL in the inner IP header is decremented by only one if the tunnel is being done as part of forwarding the datagram; otherwise the inner header TTL is not changed during encapsulation. If the resulting TTL in the inner IP header is zero, the datagram is discarded and an ICMP Time Exceeded message is returned to the sender. Therefore, an encapsulator will not encapsulate a datagram with TTL=0.
- Encapsulation of IPv6 in IPv4:
    - Utilizes IPv4 routing and properties.
    - Loses special IPv6 features.
    - Requires a hole in firewall to allow through protocol 41 (IP in IP).
- If a tunnel falls entirely within a routing domain, it will be considered as plain serial link by interior routing protocol such as RIP or OSPF. But if it lies between two routing domains it needs exterior protocols like BGP etc..
- In case of congestion in the tunnel, an ICMP Source Quench message will be issued in order to inform the previous node of the congestion.

In different types of tunneling, only de/encapsulation points are varied depending on the start and end of tunnels, however the basic idea remains the same. The three tunneling mechanisms are:

- Host-Host Tunneling
- Router-Router Tunneling
- Host-Router and vice versa Tunneling

We chose to pursue the IPv6 in IPv4 tunneling as a transition mechanism because it would be the most cost effective and can implement islands of IPv6 networks that can be connected over the existing ocean of IPv4 networks, the existing infrastructure. With time, as the islands grow, the ocean will diminish to a point that all the islands will touch, at which point it is evident that native IPv6 networks will finally reign and benefit from its new features 100%.

2.3.1 Host-to-Host Tunneling

In host to host tunneling method, encapsulation is done at source host and decapsulation is done at destination host. So the tunnel is created in between two hosts supporting both IPv4 and IPv6 stacks. So in this way encapsulated datagrams are sent through the tunnel over the IPv4 network.

In Fig. 3, it is clear that both hosts having dual stack encapsulate the packets of IPv6 in IPv4 packets and transmit over the network as an IPv4 packet utilizing all the characteristics and routing mechanisms of IPv4. With this transition mechanism, it is possible to support IPv6 simply by upgrading the end hosts protocol stacks to IPv6 while leaving the IPv4 infrastructure unchanged.

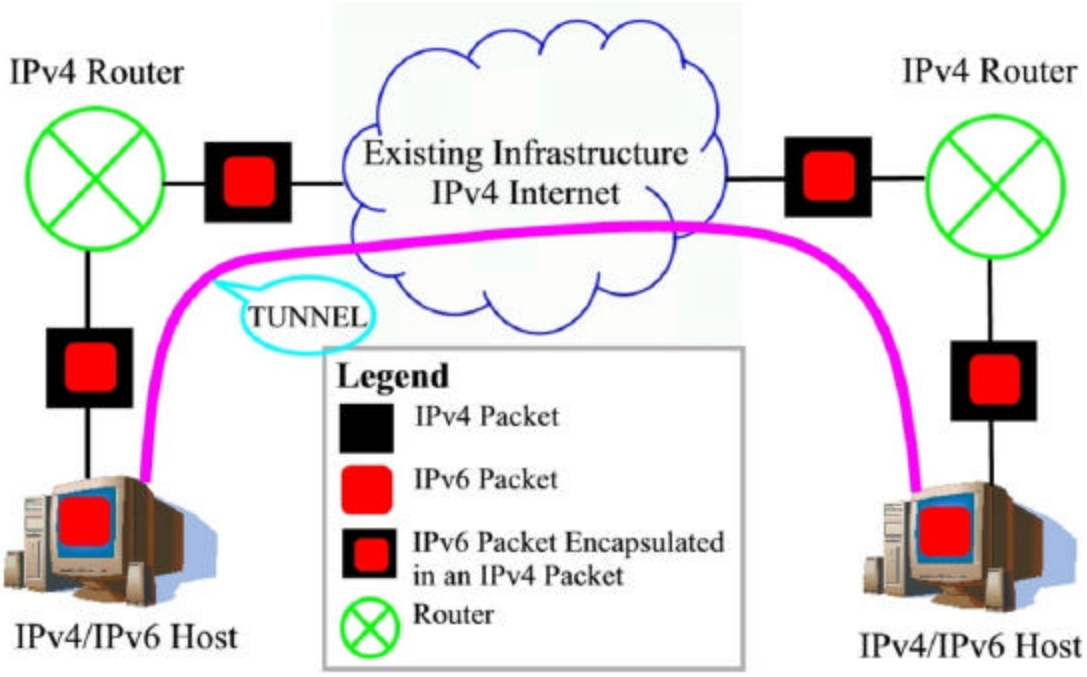

Figure 3. Host-to-Host Tunneling

2.3.2 Router-to-Router Tunneling

In router to router tunneling mechanism, encapsulation is done at edge router of originating host and decapsulation is done in the same way at edge router of destined host. The tunnel is created in between two edge routers supporting both IPv4 and IPv6 stacks. Therefore, the end hosts can support native IPv6 protocol stack while the edge routers create the tunnels and handle the encapsulation and decapsulation in order to transmit the packets over the existing IPv4 infrastructure.

Fig. 4 shows a tunnel established between two edge routers, which supports both (IPv4 / IPv6) stacks. The IPv6 datagrams are forwarded from host to edge routers while encapsulation takes place at the

router level; similarly at the other end, the reverse process takes place. In this method, both edge routers need to support dual stacks and established a tunnel prior to transmission.

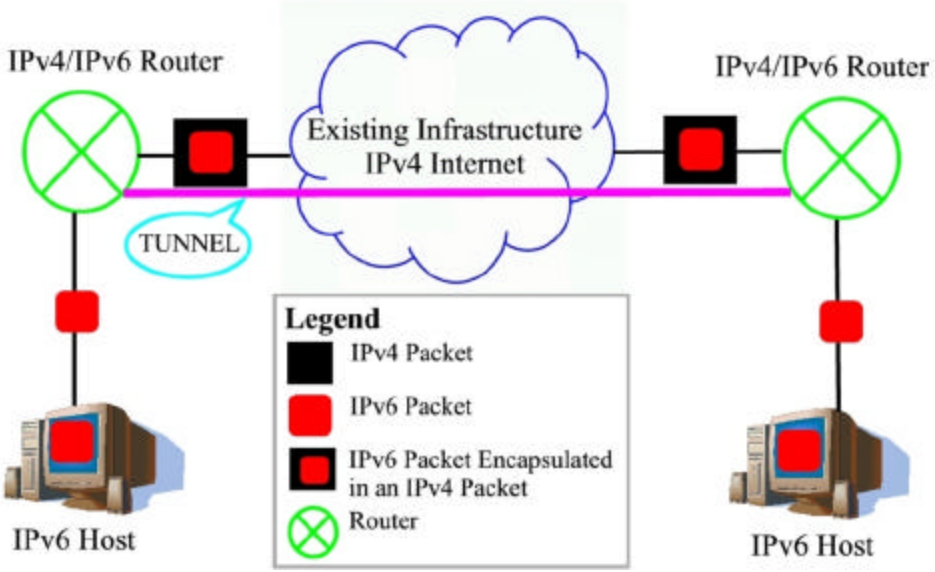

Figure 4. Router-to-Router Tunneling

2.3.3 Host-to-Router Tunneling

In host to router tunneling mechanism, encapsulation is done at originating host and decapsulation is done in the same way at edge router of destined host and vice versa. The tunnel is created in between one host and one edge router both of them supporting both IPv4 and IPv6 stacks. So in this way encapsulated datagrams are sent through the tunnel over the existing IPv4 network. The same process can happen the other way around, from one edge router to an end host.

Fig. 5 shows clearly that on one end encapsulation is taken place at the host and the other at the router; the tunnel is therefore established between the host and the router. In this method one dual stack supporting router and one dual stack supporting host is required.

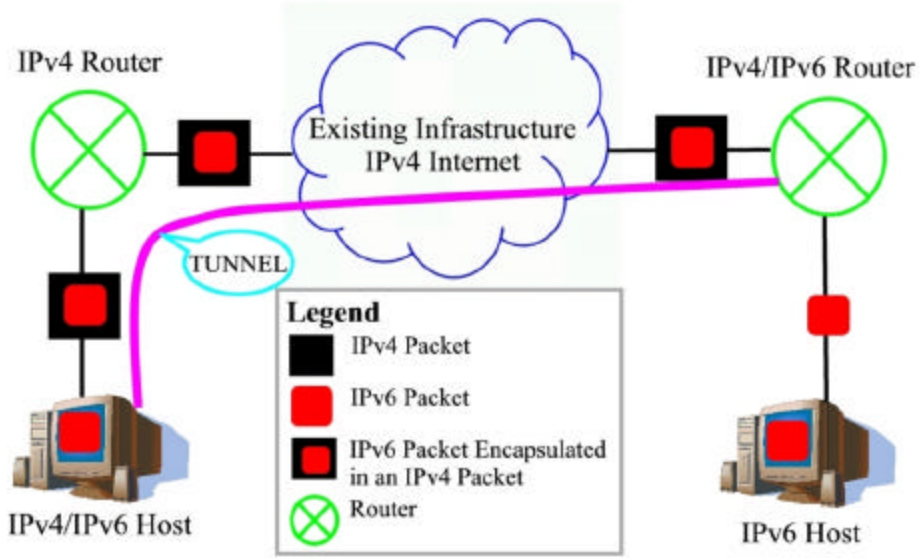

Figure 5. Host-to-Router Tunneling

3.0 Implementation

Our test-bed consisted of an Ericsson AXI 462 router that is both IPv4 and IPv6 enabled as specified in RFC 1933 [3]. The router has two separate cards, the TBC103 which is used for configuring the router, and the TBC120 with 4 different LAN cards, which can connect up 4 different networks (LANs). Some of its supported protocols are: IPv4, IPv6, RSVP, QoS, and ATM.

The workstations we used were connected directly to the router and were configured to be on 2 separate networks (192.168.xxx.xxx and 10.xxx.xxx.xxx) in order to force messages from one workstation to go through the router in order to get to its destination. The workstations were equipped with Pentium III 500 MHz processors and had 256 megabytes of RAM; they each utilized an IBM 30GB IDE hard drive and had 3COM 10/100 PCI network adapters.

Both workstations were loaded with both Windows 2000 Professional. Windows 2000 came preloaded with an IPv4 stack, however in order to get IPv6 support, an add-on package was installed. There were two choices, both written by Microsoft and they were both in Beta testing. We chose the newer release of the two, "Microsoft IPv6 Technology Preview for Windows 2000" [6] which is supported by Winsock 2 as its programming API. It was evident that Microsoft's IPv6 stack for Windows 2000 is not in production yet since it had various deficiencies. It did not seem to handle fragmentation well, and therefore we limited our test to message sizes less than the Ethernet MTU size of 1514 bytes. It also does not support IPSec yet, but that was outside of the scope of this paper and therefore is not important.

3.1 Router Configuration

The following commands were used in order to configure the Ericsson Telebit AXI 462 router. The summary of the configuration parameters is:

TBC103 Card:

LAN1:	141.217.17.50
LAN1_Ipv6:	1:1:1:1:1:1:1:1
Enabled Protocols:	IPv4, IPv6, RSVP, RIP, RIPng

Enabled RSVP support in the router. Notice that each LAN card must be enabled individually.

AXI462 % use ip routing 1
AXI462 % rsvp enable

TBC120 Card:

LAN3_Ipv4:	10.0.0.2
LAN3_Ipv6:	3:3:3:3:3:3:3:1
Enabled Protocols:	IPv4, IPv6, RSVP, RIP, RIPng
LAN4_Ipv4:	141.217.17.49
LAN4_Ipv6:	4:4:4:4:4:4:4:1
Enabled Protocols:	IPv4, IPv6, RSVP, RIP, RIPng
LAN5_Ipv4:	192.168.0.1
LAN5_Ipv6:	5:5:5:5:5:5:5:1
Enabled Protocols:	IPv4, IPv6, RSVP, RIP, RIPng
LAN6_Ipv4:	10.0.0.1
LAN6_Ipv6:	6:6:6:6:6:6:6:1
Enabled Protocols:	IPv4, IPv6, RSVP, RIP, RIPng

Configure TBC103, the configuration LAN port. Start the RIP, RIPng (RIP for IPv6), and RSVP processes for each LAN card.

AXI462 % ip routing 1
AXI462 % use ip enable 1
AXI462 % use interface lan 1
AXI462 % ip access LAN1_Ipv6 -local 1:1:1:1:1:1:1:1 -prefix 64
AXI462 % start ripng
AXI462 % use ip access lan1
AXI462 % ip rip
AXI462 % start rip

Configure TBC120; contains 4 different LAN cards, labeled LAN3, LAN4, LAN5, and LAN6 respectively. Each LAN card supports the dual stack IPv4/IPv6 and can be configured individually. We started the RIP and RIPNG (RIP for IPv6) processes for each card individually.

AXI462 % use ip routing 1
AXI462 % ip enable 3
AXI462 % interface lan 3
AXI462 % ip access LAN3_Ipv4 -local 10.0.0.2 -prefix 8 -rsvp
AXI462 % ip laninterface
AXI462 % ip rip
AXI462 % start rip
AXI462 % ip access LAN3_Ipv6 -local 3:3:3:3:3:3:3:1 -prefix 64
AXI462 % ip laninterface -secondary
AXI462 % ip ripng
AXI462 % start ripng

AXI462 % interface lan 4
AXI462 % ip access LAN4_Ipv4 -local 141.217.17.49 -prefix 24 -rsvp
AXI462 % ip laninterface
AXI462 % ip rip
AXI462 % start rip
AXI462 % ip access LAN4_Ipv6 -local 4:4:4:4:4:4:4:1 -prefix 64
AXI462 % ip laninterface -secondary
AXI462 % ip ripng
AXI462 % start ripng

XI462 % interface lan 5
AXI462 % ip access LAN5_Ipv4 -local 192.168.0.1 -prefix 24 -rsvp
AXI462 % ip laninterface
AXI462 % ip rip
AXI462 % start rip
AXI462 % ip access LAN5_Ipv6 -local 5:5:5:5:5:5:5:1 -prefix 64
AXI462 % ip laninterface -secondary
AXI462 % ip ripng
AXI462 % start ripng

AXI462 % interface lan 6
AXI462 % ip access LAN6_Ipv4 -local 10.0.0.1 -prefix 8 -rsvp
AXI462 % ip laninterface
AXI462 % ip rip
AXI462 % start rip
AXI462 % ip access LAN6_Ipv6 -local 6:6:6:6:6:6:6:1 -prefix 64
AXI462 % ip laninterface -secondary

```
AXI462 % ip ripng
AXI462 % start ripng
```

The configuration as seen by the router when we completed all configuration commands was as follows:

```
AXI462 % show ip access
use ip enable  1
ip access    lan1:  laninterface.1
      -local 141.217.17.50 -prefix 24 -State Up \
      -Lastch {2001-12-18 08:20:43} -Noofup 5
ip access LAN1_Ipv6:  laninterface.1
      -local 1:1:1:1:1:1:1:1 -secondary -State Up \
      -Lastch {2001-12-15 23:06:49} -Noofup 1
use ip enable  3
ip access   TUNNEL:      4tunnel.0
      -local 4:4:4:4:4:4:4:1 -peer 3:3:3:3:3:3:3:1 -State Up \
      -Lastch {2001-12-15 23:17:00} -Noofup 1
ip access LAN3_Ipv4:  laninterface.3
      -local 10.0.0.2 -State Up -Lastch {2001-12-15 23:04:23} -Noofup 1
ip access LAN3_Ipv6:  laninterface.3
      -local 3:3:3:3:3:3:3:1 -secondary -State Up \
      -Lastch {2001-12-15 23:06:49} -Noofup 1
ip access LAN4_Ipv4:  laninterface.4
      -local 141.217.17.49 -prefix 24 -State Up \
      -Lastch {2001-12-15 23:09:07} -Noofup 1
ip access LAN4_Ipv6:  laninterface.4
      -local 4:4:4:4:4:4:4:1 -secondary -State Up \
      -Lastch {2001-12-15 23:10:07} -Noofup 1
ip access LAN5_Ipv4:  laninterface.5
      -local 192.168.0.1 -State Down -Lastch {2001-12-18 21:03:42} \
      -Noofup 5
ip access LAN5_Ipv6:  laninterface.5
      -local 5:5:5:5:5:5:5:1 -secondary -State Down \
      -Lastch {2001-12-18 21:03:42} -Noofup 5
ip access LAN6_Ipv4:  laninterface.6
      -local 10.0.0.1 -State Down -Lastch {2001-12-18 21:40:44} -Noofup 3
ip access LAN6_Ipv6:  laninterface.6
      -local 6:6:6:6:6:6:6:1 -secondary -State Down \
      -Lastch {2001-12-18 21:40:44} -Noofup 3
```

3.2 Measurement Procedures

Our metrics of evaluation were: throughput and latency. All the performance measurement software was written in C++.

The majority of the tests were done for a period of about 60 seconds, which netted about 50000 packets to about 1000000 packets, depending on the size of the packets sent. All tests were done using various packet sizes ranging from 64 bytes to 1024 bytes. Each test was repeated three times in order to rule out any inconsistencies.

3.2.1 Throughput

Throughput is a very clear representation of the real overhead incurred by the header information. A network link only has the total bandwidth capacity to transmit its packets which include all the headers for the different layers and the final payload of usable data. Obviously, no system can ever achieve throughputs of 100% of the bandwidth due to the overhead of header information. For example, using IPv4 UDP, in a best case scenario, the system would only achieve 97.2% capacity of the bandwidth (see Table 1 for details). Throughput was calculated by sending XX number of packets of YY bytes from a client to a server. At the beginning of the test, the time would recorded; at the end of the test, again the time would be noted. The two timestamps, which have a microsecond resolution, would be subtracted from each other, and what remained would be the duration of the test in microseconds. Since we knew the size of the messages we were sending (YY) and the number of messages we eventually sent (XX), we knew that we sent XX*YY number of bytes over the change in microseconds, and therefore were could translate the result into Mbit/s.

3.2.2 Latency

Latency, or better know as RTT (round trip time), is very important since many applications are sensitive to any kind of delays. Having better latency could mean that the protocol would perform better for real time applications such as video or audio. Latency was calculated by sending a message of XX bytes from a client to a server; upon the receipt of the message, the server sent back the same message back to the original client; when the client received its message back, the whole process would start all over again reiterating the same process for YY number of times. At the end of the test, we knew that we had YY iterations with a length of so many microseconds, and therefore could derive the RTT by dividing the duration by YY.

4.0 Performance Results

For Fig. 1 through Fig. 4, we will use the following consistent conventions as explained in the legends of each figure. The x-axis is the packet size in the corresponding experiment, while the y-axis represents the measured metric. For each test, we have two figures denoting the results for both TCP and UDP sockets.

4.1 Throughput

4.1.1 TCP

As Fig. 6 indicates, it can be clearly seen that each layer of complexity adds additional overhead over the entire packet size spectrum. Although the IPv4 and IPv6 stacks present similar trends, IPv6 incurs an additional 5% to 18% overhead ranging from the larger packet sizes to the smaller ones. On top of the already incurred overhead of IPv6, Host-Host tunneling has an even more drastic performance hit, as high as 50% for big packet sizes. Even more, the CPU usage was a clear indication of the load each protocol was placing on the end hosts. There were differences of 10 to 25% in CPU usage when comparing the three different protocols, all consistent to the below graph; the higher the throughput, the lower the CPU usage. We are positive that the router experiences similar behaviors in the CPU load, however, we have no way to verify our assumptions.

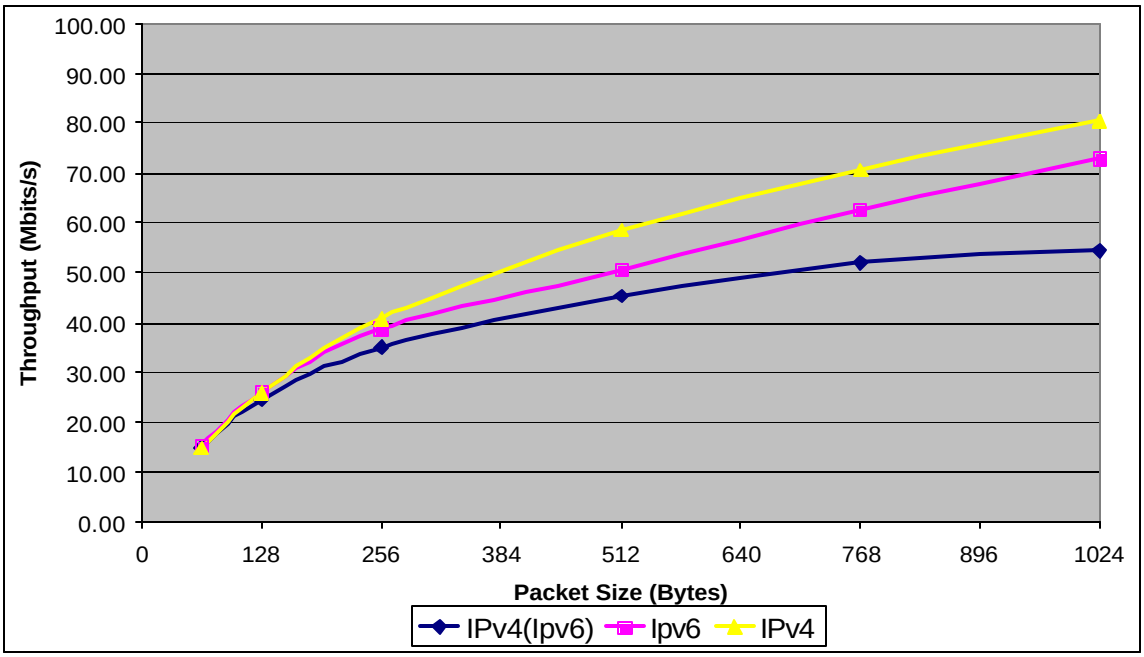

Figure 6: TCP throughput results for IPv4 and IPv6 with packets ranging from 64 to 1024 Bytes

4.1.2 UDP

As Fig. 7 indicates, the results are comparable to those of TCP, except that IPv6 and Host-Host tunneling level off while IPv4 maintains its upward trend. We suspect that the IPv6 has some memory buffer allocation problems which caused it to level off at about 50 Mbit/s as it did. The CPU usage was also very similar. The numbers reflected below are for the sending host; the receiving host was generally dropping very few packets (1~2%), so therefore we did not display the receiving throughput.

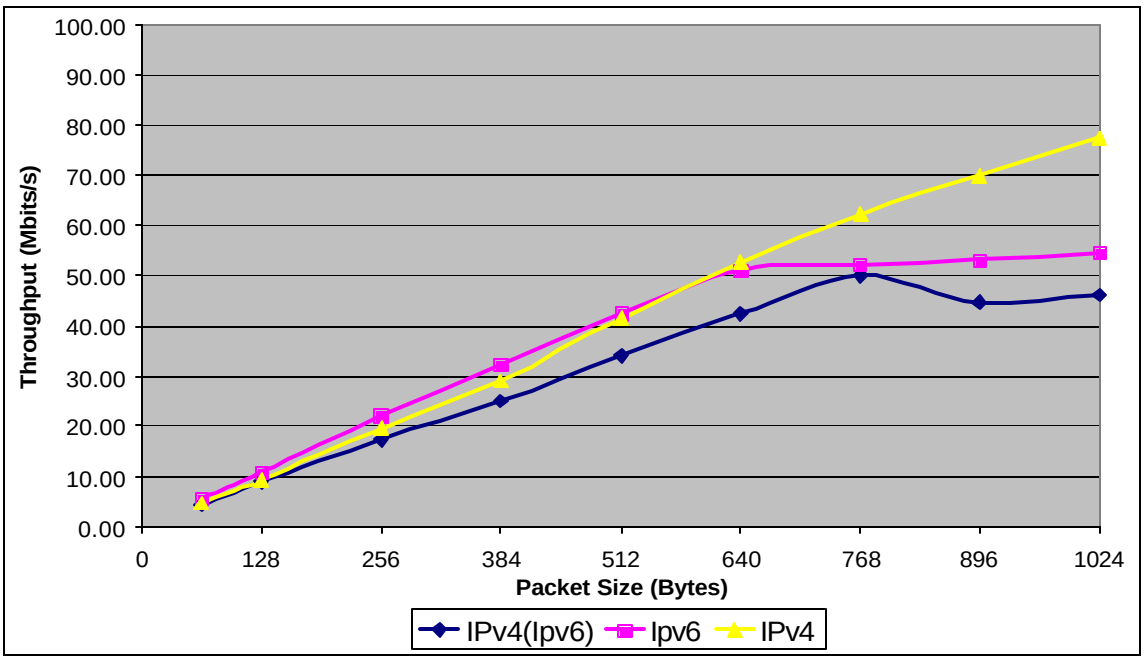

Figure 7: UDP throughput results for IPv4 and IPv6 with packets ranging from 64 to 1024 Bytes

4.2 Latency

4.2.1 TCP

As Fig. 8 indicates, all three methods yield similar trends. IPv6 incurred an additional 2.5% to 15% overhead above IPv4. Host-Host tunneling had a 13% to 35% overhead on top of IPv4. The CPU usage was consistently lower by about 10% between each respective protocol; the higher the RTT, the higher the CPU utilization.

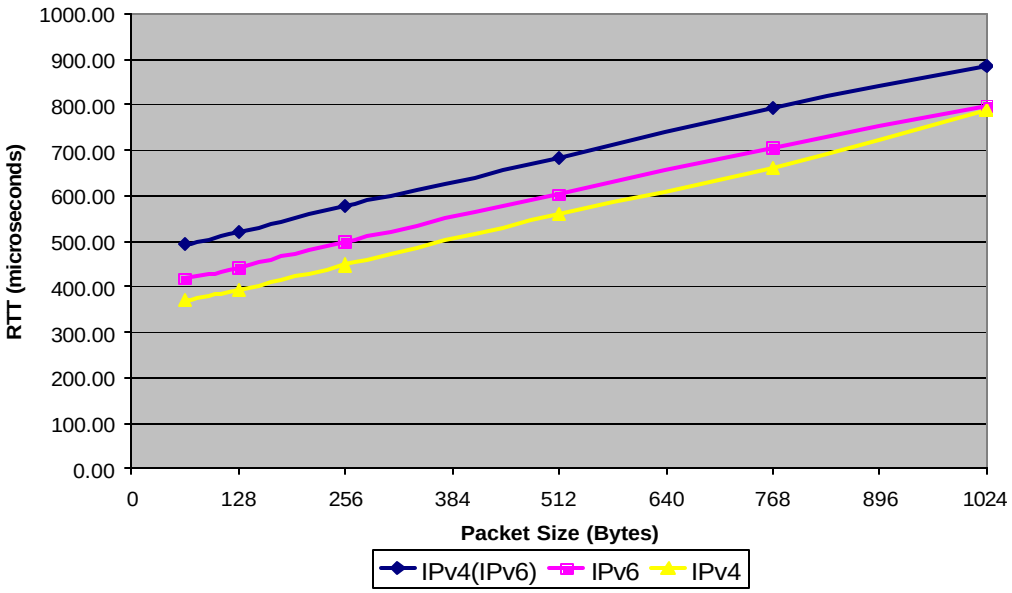

Figure 8: TCP latency results for both IPv4 and IPv6 with packets ranging from 64 to 1024 Bytes

4.2.2 UDP

As Fig. 4 indicates, all three methods yield similar trends. IPv6 incurred an additional 12.5% to 20% overhead above the IPv4 stack. Host-Host tunneling had a 18% to 35% overhead on top of IPv4.

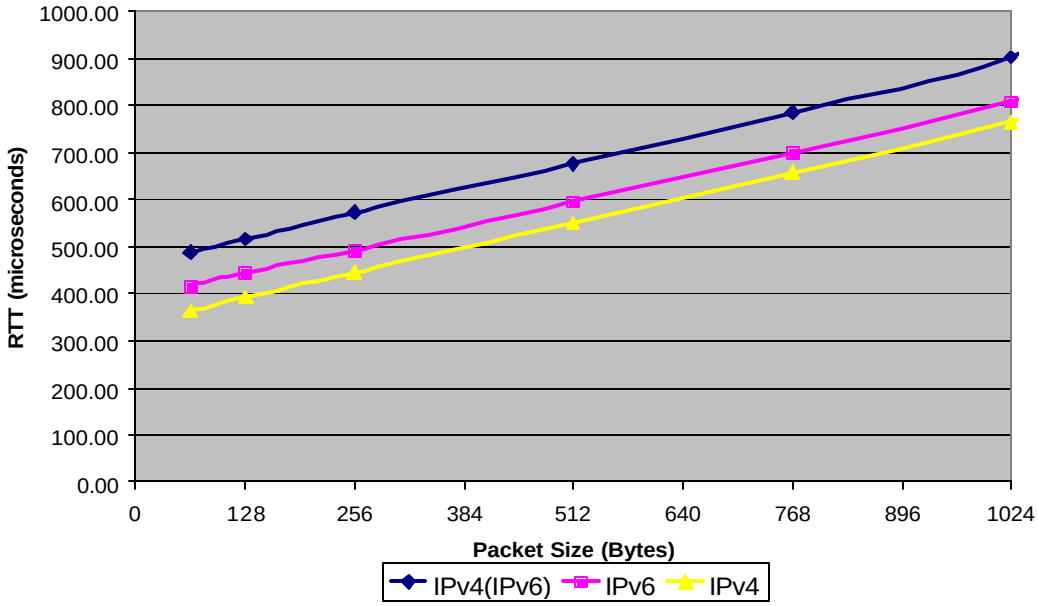

Figure 4: UDP latency results for both IPv4 and IPv6 with packets ranging from 64 to 1024 Bytes

5.0 Conclusion

Through our work, we presented an unbiased empirical performance evaluation between IPv4, IPv6, and Host-to-Host Tunneling running over Windows 2000. For every level of complexity (IPv4, IPv6, 6over4), the performance overhead was always increasing. Note that the anticipated 1.5% overhead calculated in Section 2 was matched with a 5% to 36% overhead incurred in actual testing. As IPv6 is still maturing, perhaps it is just a matter of time until its performance will finally reflect its theoretical counterpart.

In the near future, we plan on extending the work be began here to Solaris 8.0 in order to compare two different OS architectures and two different implementations of the protocol stacks. IPv6 also supports prioritizing packets, which might be an easy way to offer a lighter version of QoS without specifying any requirements.

The real value of our work lies in the potential of IPv6. In this paper, we only covered the most basic features of IPv6, and we believe that as we pursue some of the new QoS features and perhaps mobility support, it will become more clear the superiority of IPv6 over IP4. According to our evaluation, IPv6 has a performance deficit when utilizing traditional data streams, but as multimedia content is becoming more abundant in the Internet, only an in-depth evaluation of new emerging applications will net the real performance gain of IPv6.

6.0 References

A. S. Tanenbaum, Computer Networks, Third Edition, Prentice Hall Inc., 1996, pp. 276-287.

S. Deering, R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," Request for Comments 1883, Internet Engineering Task Force, December 1995.

S. Deering, R. Hinden, "IP Version 6 Addressing Architecture," Request for Comments 1884, Internet Engineering Task Force, December 1995.

S. Thomson, T. Narten, "IPv6 Stateless Address Autoconfiguration," Request for Comments 1971, Internet Engineering Task Force, August 1996.

T. Narten, E. Nordmark, W. Simpson, "Neighbor Discovery for IP Version 6 (IPv6)," Request for Comments 1970, Internet Engineering Task Force, August 1996.

C. Huitema, "IPv6, The New Internet Protocol, Second Edition," Prentice Hall Inc., 1997, pp. 197-221.

R. Gilligan, E. Nordmark, "Transition Mechanisms for IPv6 Hosts and Routers," Request for Comments 1933, Internet Engineering Task Force, April 1996.

G. Tsirtsis, P. Srisuresh, "Network Address Translation - Protocol Translation (NAT-PT)," Request for Comments 2766, Internet Engineering Task Force, February 2000.

A. Durand, P. Fasano, I. Guardini, D. Lento, "IPv6 Tunnel Broker, " Request for Comments 3053, Internet Engineering Task Force, January 2001.

B. Carpenter, K. Moore, "Connection of IPv6 Domains via IPv4 Clouds," Request for Comments 3056, Internet Engineering Task Force, February 2001.

A. Conta, S. Deering, "Generic Packet Tunneling in IPv6 Specification," Request for Comments 2473, Internet Engineering Task Force, December 1998

B. Carpenter, C. Jung, "Transmission of IPv6 over IPv4 Domains without Explicit Tunnels," Request for Comments 2529, Internet Engineering Task Force, March 1999.

Ericsson, "AXI460 Documentation version 1.3", 2000.

Fiuczynski, Marc E et. Al. "The Design and Implementation of an IPv6/IPv4 Network Adress and Protocol Translator". 1998.

Karuppiah, Ettikan Kandasamy, et al. "Application Performance Analysis in Transition Mechanism from IPv4 to IPv6". 2000.

Draves, Richard P., et al. "Implementing IPv6 for Windows NT". Proceedings of the 2nd USENIX Windows NT Symposium, Seatle, WA, August 3-4, 1998.

Information Sciences Institute, University of Southern California, "Internet Protocol," Request for Comments 791, Internet Engineering Task Force, September 1981

S. Bradner, A. Mankin, "IP: Next Generation (IPng) White Paper Solicitation," Request for Comments 1550, Internet Engineering Task Force, December 1993

R. Gilligan, E. Nordmark, "Transition Mechanisms for IPv6 Hosts and Routers," Request for Comments 1933, Internet Engineering Task Force, April 1996

P. Srisuresh, M. Holdrege, "IP Network Address Translator (NAT) Terminology and Considerations," Request for Comments 2663, Internet Engineering Task Force, August 1999

C. Huitema, "The H Ratio for Address Assignment Efficiency," Request for Comments 1715, Internet Engineering Task Force, November 1994

Microsoft Corporation, "Microsoft IPv6 Technology Preview for Windows 2000," December 12, 2000, http://www.microsoft.com/windows2000/technologies/communications/ipv6/default.asp