Project 3
IPv6 Performance Results

Author: **Ioan Raicu**
SS#: **171-72-8942**
Email: **iraicu@cs.wayne.edu**
Course: **CSC6991**

Document Created: 2/5/2000
Last Date Modified: 9/17/2001

Table of Contents

1.0 Proposal

Due to the freedom we were granted in choosing a topic for this project, I decided to elect a topic that flows in the same direction as my research areas in which I am currently involved in the HSNL. The project will have to do with an implementation of IPv6 and obtaining some performance results.

According to what I have been reading, Microsoft has assured itself a leading role in the development of IPv6, and thus I have started my work from their documentation and am slowly making my way out to other work related to IPv6. 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" will only work 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. I will give a full-blown specification of the utilities provided by Microsoft in a later section.

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. Microsoft even included a utility called "checkv4.exe" which will scan IPv4 source code and give recommendations on where the code should be changed in order to support IPv6.

I will create an application using C++ and utilizing Winsock2 to make a program that will send and receive IPv6 packets using both UDP and TCP protocols. I will then test the performance of Ipv6 vs. Ipv4 under this simple implementation. The later part will involve utilizing the priority field in the Ipv6 header, which will hopefully be understood by the Ericsson Telebit Router AXI 462, which we are expecting very soon. I will be designing the test application for the Windows platform, while Davis Ford will perform some similar tests on Solaris 8.0 for both the Intel and Sparc.

Through this project, I hope to better my skills with IPv6 socket programming and gain valuable information regarding performance differences between Ipv4 vs. Ipv6 and Windows vs. Solaris. The beauty of this project is its comprehensive coverage of different protocols (TCP vs. UDP), different generation of protocols (IPv4 vs. IPv6), different OSs (NT 4.0, Windows 2000, Solaris 8.0), and different hardware architectures (Intel vs. Sparc).

2.0 Overview

The following sections will go into much detail describing the fundamental differences between various testing criteria. This section is very important for readers who are new to the diverse network protocols that we used to come up with the performance results. Keep in mind that this is only an overview, and thus for a complete description for any of the described topics, further resources will have to be considered. I have included my source for the information, so logically following up with my sources should be the first place anyone should look for more information.

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.

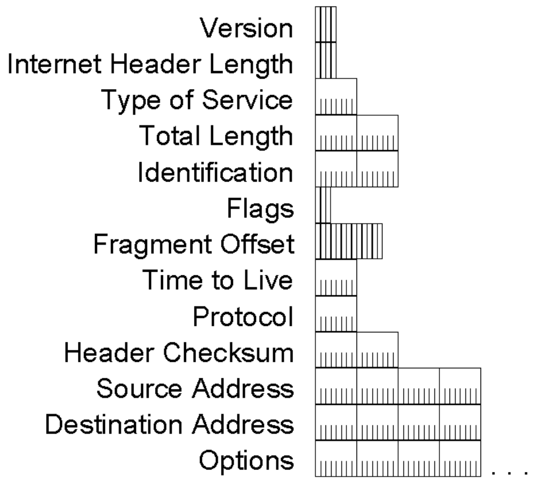


Figure 1   The IPv4 header

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.

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

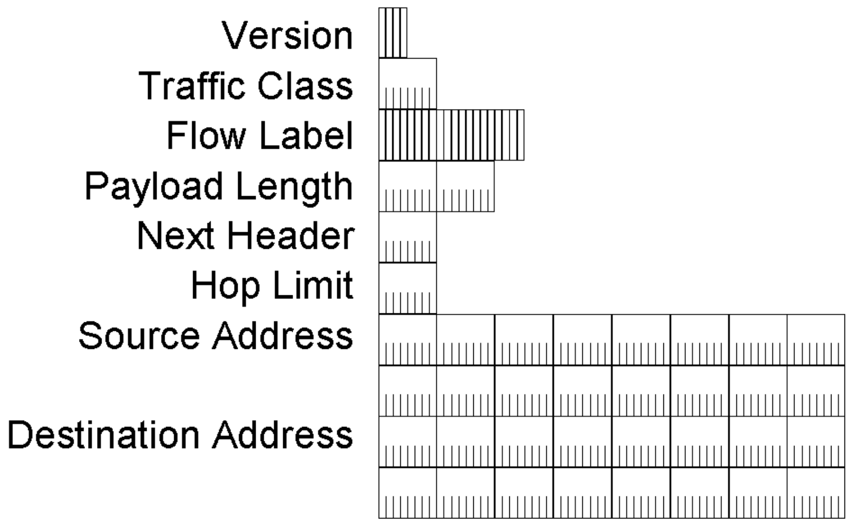


Figure 2 The IPv6 header

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.
    - **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.

2.2.2.1 Microsoft Research IPv6 Release 1.4 [14]

2.2.2.2 Microsoft IPv6 Technology Preview for Windows 2000 [15]

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 TCP	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. For more information, see “IPv6 Addressing.”
IPSec support is optional.	IPSec support is required. For more information, see “IPv6 Header.”
No identification of payload for QoS handling by routers is present within the IPv4 header.	Payload identification for QoS handling by routers is included in the IPv6 header using the Flow Label field. For more information, see “IPv6 Header.”
Fragmentation is supported at both routers and the sending host.	Fragmentation is not supported at routers. It is only supported at the sending host. For more information, see “IPv6 Header.”
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. For more information, see “IPv6 Header.”
Header includes options.	All optional data is moved to IPv6 extension headers. For more information, see “IPv6 Header.”
Address Resolution Protocol (ARP) uses broadcast ARP Request frames to resolve an IPv4 address to a link layer address.	ARP Request frames are replaced with multicast Neighbor Solicitation messages. For more information, see “Neighbor Discovery.”
Internet Group Management Protocol (IGMP) is used to manage local subnet group membership.	IGMP is replaced with Multicast Listener Discovery (MLD) messages. For more information, see “Multicast Listener Discovery.”
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 messages and is required. For more information, see “Neighbor Discovery.”
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. For more information, see “Multicast IPv6 Addresses.”
Must be configured either manually or through DHCP.	Does not require manual configuration or DHCP. For more information, see “Address Autoconfiguration.”
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 Domain Name System (DNS) to map host names to IPv6 addresses. For more information, see “IPv6 and DNS.”

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. For more information, see "IPv6 and DNS."

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 Operating Systems

2.3.1 Microsoft Windows NT 4.0

2.3.2 Microsoft Windows 2000

2.3.3 Solaris 8.0

2.4 Hardware Architectures

2.4.1 Intel

2.4.2 Sparc

3.0 Implementation

I used several Microsoft examples in order to get a working test program to measure various performance metrics. Among these metrics were throughput, round trip time (RTT), and delay variance. The RTT was best in finding the real overhead of IPv4 over IPv6 due to the fact that it was independent of the underlying implementation of the way the protocol implemented its send and receive mechanisms. For example, in TCP, I was able to achieve the same performance in terms of throughput using 64 B packets as I was able to achieve with 64 KB packets. However, in UDP, the performance decreased along with packet size which was the expected behaviour. This was not because TCP has less overhead on small packets than UDP does, but because TCP has an implementation option in which it can wait to fill the buffer before it sends the buffer, which essentially leads to always sending full buffers, which means that if the buffer is set large enough, it is unimportant that your packets are small because they are all sent together in one buffer. This is what yielded my awesome performance in TCP with very small packets, but was unrelated to the overhead of the protocol.

Therefore, I only concentrated on obtaining RTT for as many scenarios as I could, gathering data from packet transfers from 64 B to 64 KB under TCP, UDP, IPv4, IPv6, Windows NT 4.0, Windows 2000, etc… I experimented with socket buffers of various size, but came to the conclusion that in today's world of high speed networks in the Gbits/s bandwidth, larger buffers would be preferred over smaller ones. I used 60 KB buffers for all be displayed results in section 4.

My hardware configuration was the following:

Machine 1:

Name:	SZ06
IP Address:	192.168.0.100
Processor:	PIII 500
RAM:	256 MB SDRAM PC100
NIC:	3COM 10/100 XL (3C905C-TX)
OS:	Windows NT 4.0 & Windows 2000

Machine 2:

Name:	SZ07
IP Address:	10.0.0.100
Processor:	PIII 500
RAM:	256 MB SDRAM PC100
NIC:	3COM 10/100 XL (3C905C-TX)
OS:	Windows NT 4.0 & Windows 2000

Routers:

- IBM 2216-400 (10 Mbit/s)
- Ericsson AXI 462 (100 Mbits/s)

The RTT was obtained by sending having a client and a receiver on two different machines hooked up to the same router, running on different networks. The client would send a packet of a specific size to the server which in turn would send the same packet back of the same size. When the client would receive the packet back, it would proceed with the next packet and the process would start all over again. The first time the client would send a packet, it would get the starting timestamp and store it in a variable. When the test completed, after X amount of iterations, it would get the end timestamp. The difference between the start and end would be the total time it took to complete the test. The time has a granularity of microseconds, and thus is very accurate. The RTT would then be calculated by taking the total time and divided by the number of packets sent. The number of iterations that the client and server had to complete for any given test depended on the packet size. I was aiming for the tests to generally last between 30 to 60 seconds, so the packet numbers ranged from 1000 packets (for 64 KB packets) to about 1000000 packets (for 64 byte packets). Obviously, these numbers were lower for the IBM router due to its 10 Mbit/s connection.

Overall, the tests I performed give a general overview, but I believe they need to be performed several times, such as 10 times each test, with the lowest and highest values dropped, and averaging the remaining 8 values. I do not expect the results that has significant differences to change much, but when things were real close, within a few percent or so, it could make enough difference to change the outlook.

3.1 Data Structures

The structures that hold the information needed to transmit packets over a network are a little bit different when comparing IPv4 with IPv6. At the moment, the difference is entirely dependent upon the protocol stack, since there is no set standard yet due to the early stages in the deployment phase of IPv6.
For IPv4, the structure is sockaddr_in. The Windows Sockets sockaddr structure varies depending on the protocol selected.

```
struct sockaddr
{
        u_short    sa_family;
        char       sa_data[14];
};
```

In Windows Sockets 2, the name parameter is not strictly interpreted as a pointer to a sockaddr structure. It is presented in this manner for Windows Sockets compatibility. The actual structure is interpreted differently in the context of different address families. The only requirements are that the first u_short is the address family and the total size of the memory buffer in bytes is namelen. The size of the sockaddr_in is 32 bytes.

```
struct sockaddr_in
{
    short   sin_family;
    u_short sin_port;
    struct  in_addr sin_addr;
    char    sin_zero[8];
};
```

For IPv6, the structure is sockaddr_storage. The SOCKADDR_STORAGE structure stores socket address information. Since the SOCKADDR_STORAGE structure is sufficiently large to store IPv4 or IPv6 address information, or others, its use promotes protocol-family and protocol-version independence, and simplifies cross-platform development. Use the SOCKADDR_STORAGE structure in place of the sockaddr structure.

```
struct sockaddr_storage
{
        short ss_family;                    // Address family.
        char __ss_pad1[_SS_PAD1SIZE];  // 6 byte pad, this is to make
                                            // implementation specific pad up
                                            // to alignment field that follows
                                            // explicit in the data structure.
        __int64 __ss_align;                 // Field to force desired
                                            // structure.
        char __ss_pad2[_SS_PAD2SIZE];  // 112 byte pad to achieve desired
                                            // size;
                                            // _SS_MAXSIZE value minus size of
                                            // ss_family, __ss_pad1, and
                                            // __ss_align fields is 112.
}
```

Members definition

- **ss_family** – Address family of the socket, such as AF_INET.
- **ss_pad1** – Reserved. Defined as a 48-bit pad that ensures SOCKADDR_STORAGE achieves 64-bit alignment.
- **ss_align** – Reserved. Used by the compiler to align the structure.
- **ss_pad2** – Reserved. Used by the compiler to align the structure.

The first field of the SOCKADDR_STORAGE structure is isomorphic with the sockaddr structure to enable simplified transition from sockaddr to SOCKADDR_STORAGE. The total size of the sockaddr_storage is 128 bytes.

4.0 Performance Results

In the following few pages, there will be a series of graphs plotting the performance results of Windows NT 4.0 Workstation (Service Pack 6), Windows 2000 Professional (Service Pack 1), and Solaris 8.0.

Throughout these results, I will be using the following methodology, which is also clearly marked on each graph. The top title of each graph will describe the test that was performed. For example, for graph 1, the title “Ericsson 100Mbit/s Router, TCP – IPv4 vs. IPv6 – RTT (64 bytes ~ 65536 bytes) – 60 KB Buffer”, can be decomposed as follows:

Ericsson	– the router used; could be either Ericsson with an 100 Mbit/s connection or IBM with a 10 Mbit/s connection
TCP	– protocol used; could be either TCP or UDP
IPv4 vs. IPv6	– indicates that results will show both IPv4 and IPv6 results head-to-head
RTT	– indicates that the test performed the RTT (round trip time) for the specified packet sizes
64 bytes ~ 65536 bytes	– indicates the range of packet sizes tested in the particular graph; possible choices are 64 bytes to 65536 bytes
60 KB Buffer	– buffer allocated using setsockopt(); the tests I performed have been with either the Windows default of 8KB, or the larger variation of 60 KB

The different scales mean the following:

left	– Round trip time (RTT) measured in microseconds
bottom	– packet size measured in bytes
right	– overhead of IPv4 vs. IPv6; percentage difference between v4 and v6 RTT

The legend at the bottom of each graph also describes the different data found on each graph, but I will explain them once more for clarification.

Lines with squares	– Windows NT 4.0 results
Lines with rhombuses	– Windows 2000 results
Lines with no shape	– linear averages of specified data
Purple lines	– RTT for IPv4 under Windows NT 4.0
Light blue lines	– RTT for IPv6 under Windows NT 4.0
Red lines	– RTT for IPv4 under Windows 2000
Blue lines	– RTT for IPv6 under Windows 2000
Black lines	– % difference between IPv4 and IPv6 under Win NT4
Light green lines	– % difference between IPv4 and IPv6 under Win2000
Dark green lines	– linear average of % difference between IPv4 and IPv6 under Win NT4
Grey lines	– linear average of % difference between IPv4 and IPv6 under Windows 2000

Once again, the above information can be found in each graph, but in a more abbreviated form. Once the pattern is established and learned, it will be fairly easy to understand the graphs.

4.1 Windows NT 4.0 vs. Windows 2000 OS

According to the results obtained in table 1, it seems that Windows 2000 is more optimized for larger packets while Windows NT 4.0 is more optimized for smaller packets. Table 4 shows the relative differences between the RTT for TCP under Windows 2000 when compared to Windows NT 4.0. Be aware that the percentages indicate how much shorter the W2K RTTs were in comparison to NT4 RTT. The data of table 1 was obtained using graphs 1 through 4, so see those for more details.

Although it is clear of the behavior difference between these two OS using TCP from both the obtained data and logical sense, more tests must be performed in order to come to a certain conclusion. The reason I am saying that these results make sense is that networks have evolved greatly since the creation of Windows NT 4.0 back in 1996. My guess is that Microsoft realized that as network bandwidth is increasing for networks everywhere, optimizing their protocol stack for larger packets was the smart thing to do. Once again, this theory needs more investigation in order to certify its validity.

*Windows 2000 vs. Windows NT 4.0 RTT Performance Gain								
Packet Size (bytes)	**64**	**512**	**1024**	**2048**	**4096**	**8192**	**16384**	**65536**
IPv4	-3.72%	-3.09%	-3.01%	-2.01%	0.13%	1.39%	3.71%	2.41%
IPv6	-7.29%	-5.79%	-5.06%	-1.86%	3.36%	1.87%	3.62%	2.31%

Table 4. Performance tests through Ericsson Router using TCP
** the higher the value, the better the performance of Windows 2000 compared to Windows NT 4.0*

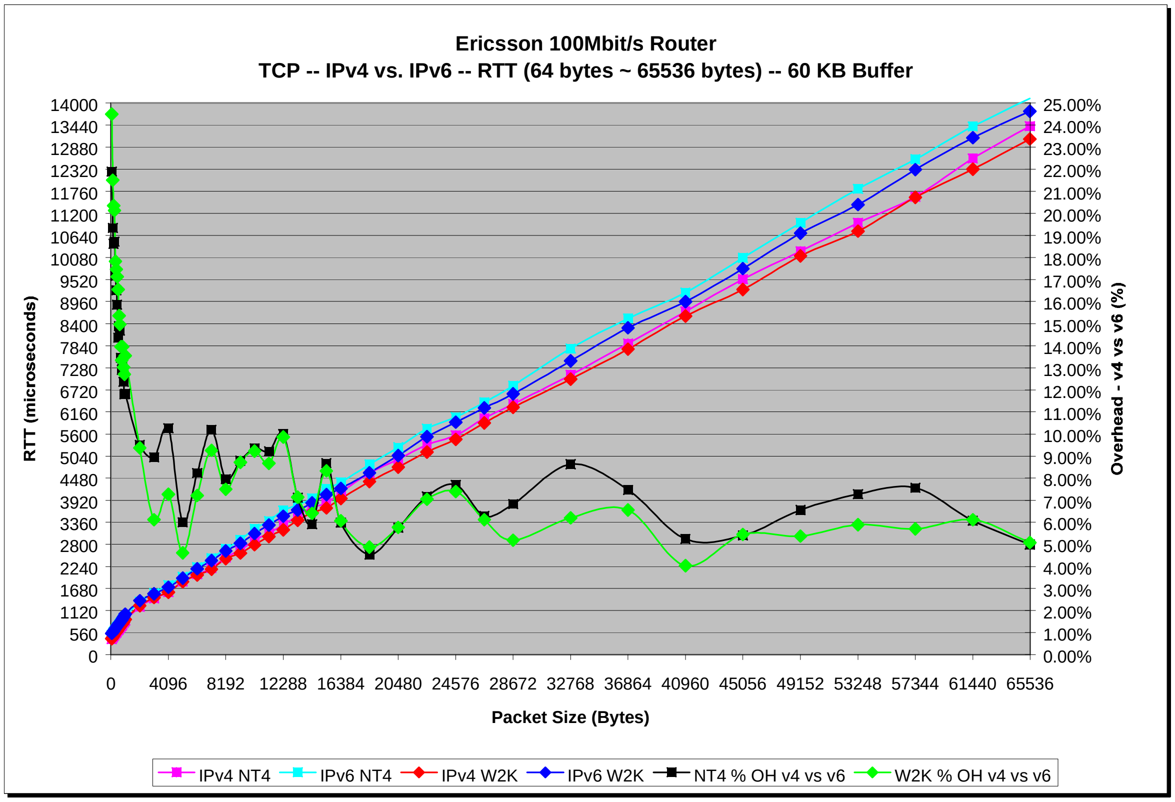
4.1.1 IPv4 vs. IPv6 Protocol

4.1.1.1 TCP Protocol

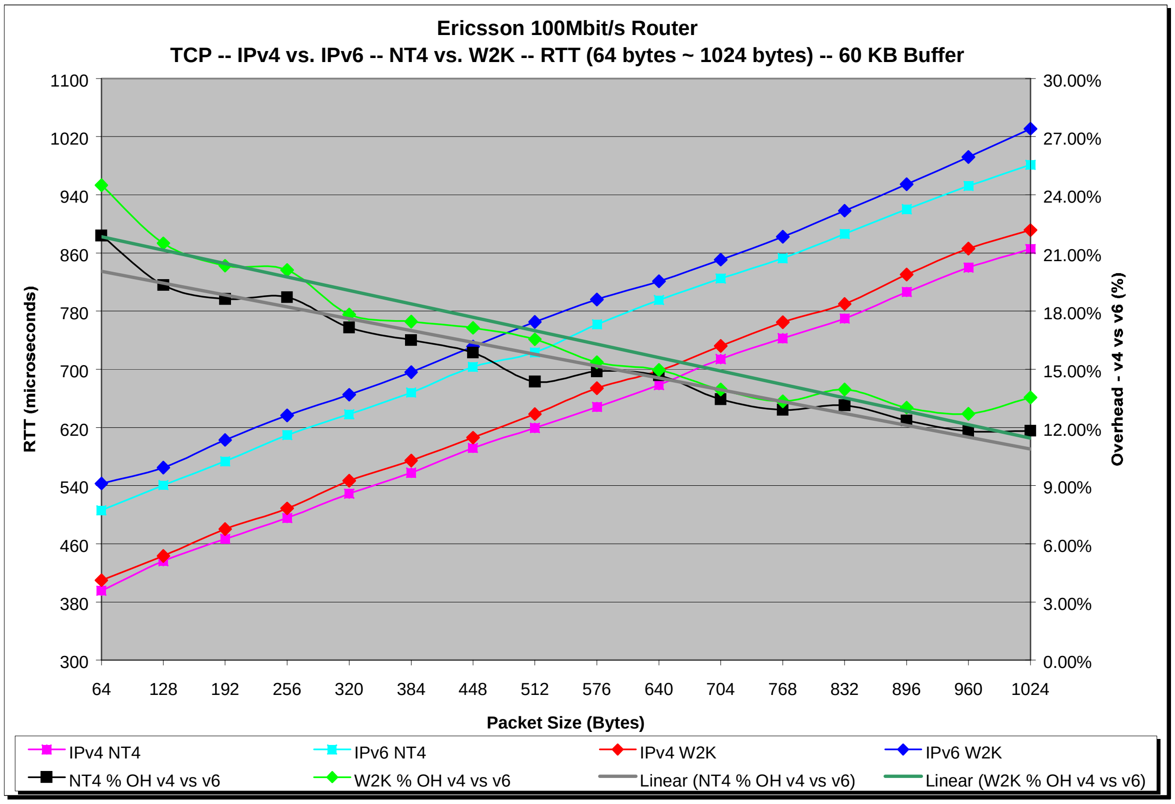
4.1.1.1.1 Ericsson 100Mbit/s Router

Below, you will find the experimental results performed on the Ericsson Router with a 100 Mbit/s connection running TCP under both Windows NT 4.0 and Windows 2000.

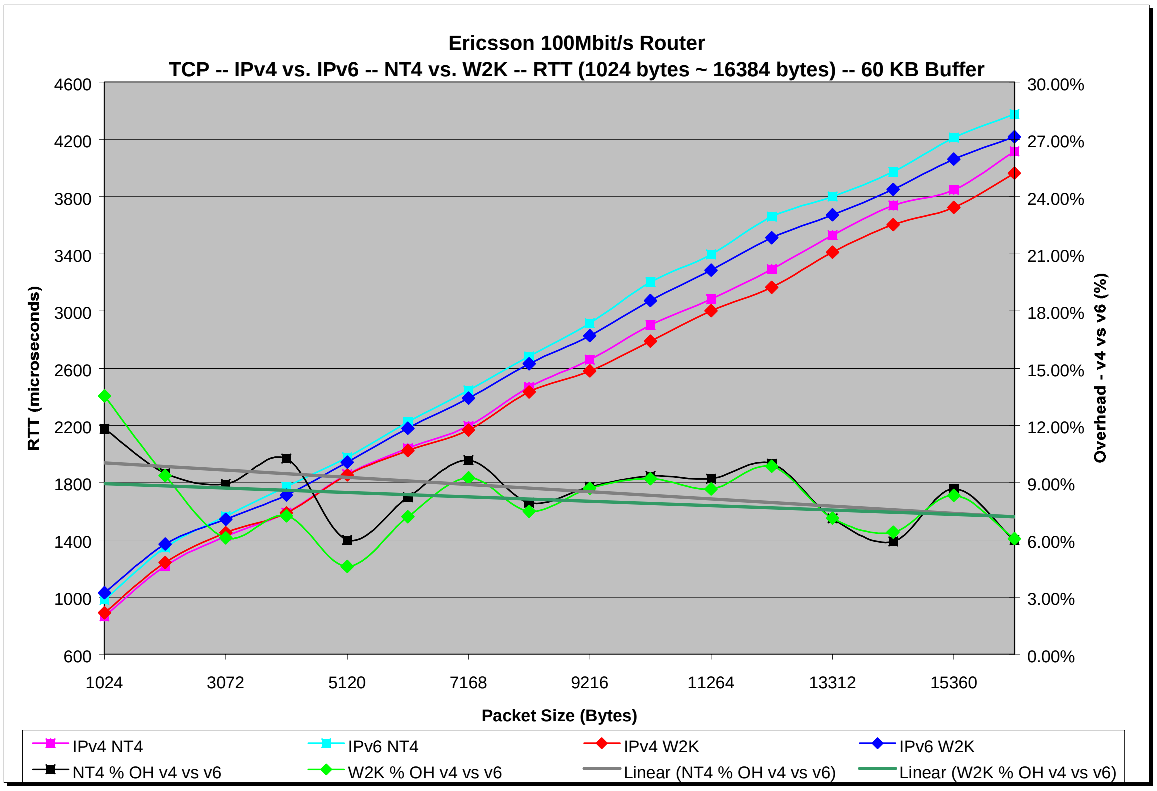
As you will notice in graphs 1 through 4, IPv4 outperforms IPv6 in both Windows NT 4.0 and Windows 2000 anywhere from about 4% when using large packets (64KB) to about 25% when using small packets (64B). In reality, because of the actual overhead of the larger IP header in IPv6, there should only be about 2% difference when using large packets and about 32% difference when using small packets. More tests need to be performed to conclude that the experimental results obtained are valid and that they were not a fluke.



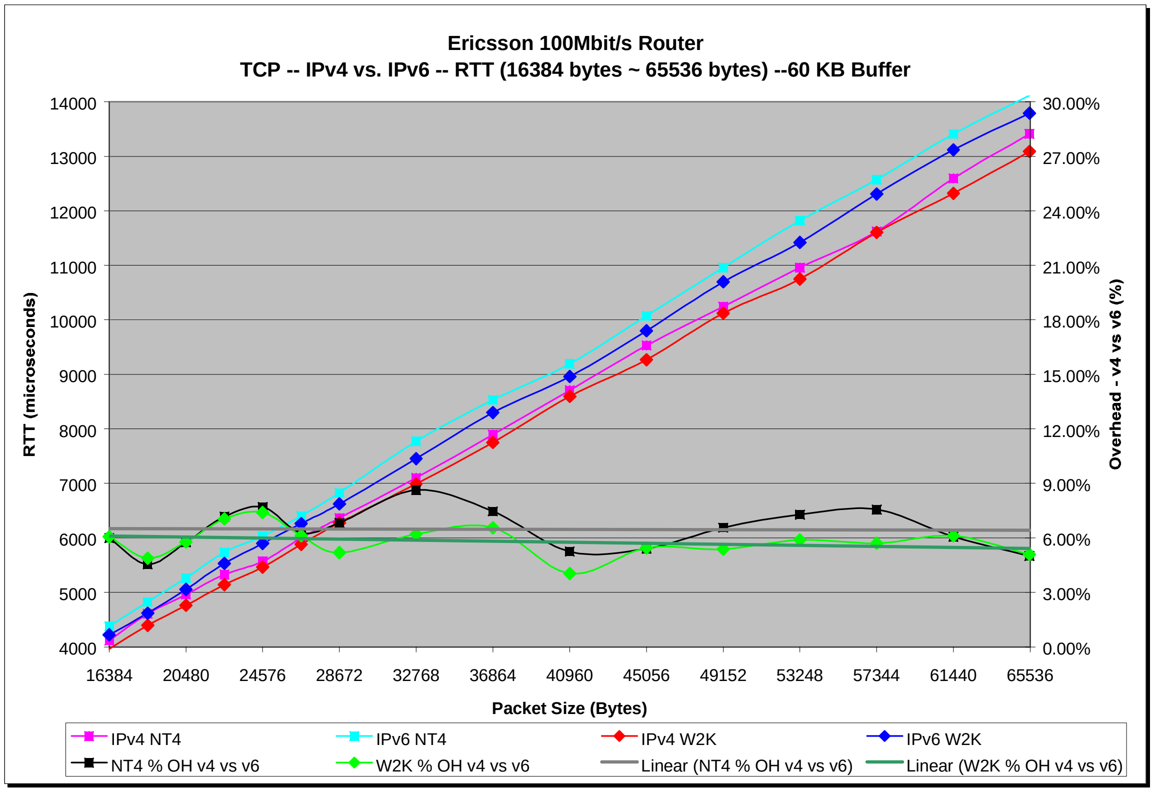
Graph 1. Ericsson Router TCP NT4 and W2K Test# 1~3



Graph 2. Ericsson Router TCP NT4 and W2K Test# 1



Graph 3. Ericsson Router TCP NT4 and W2K Test# 2

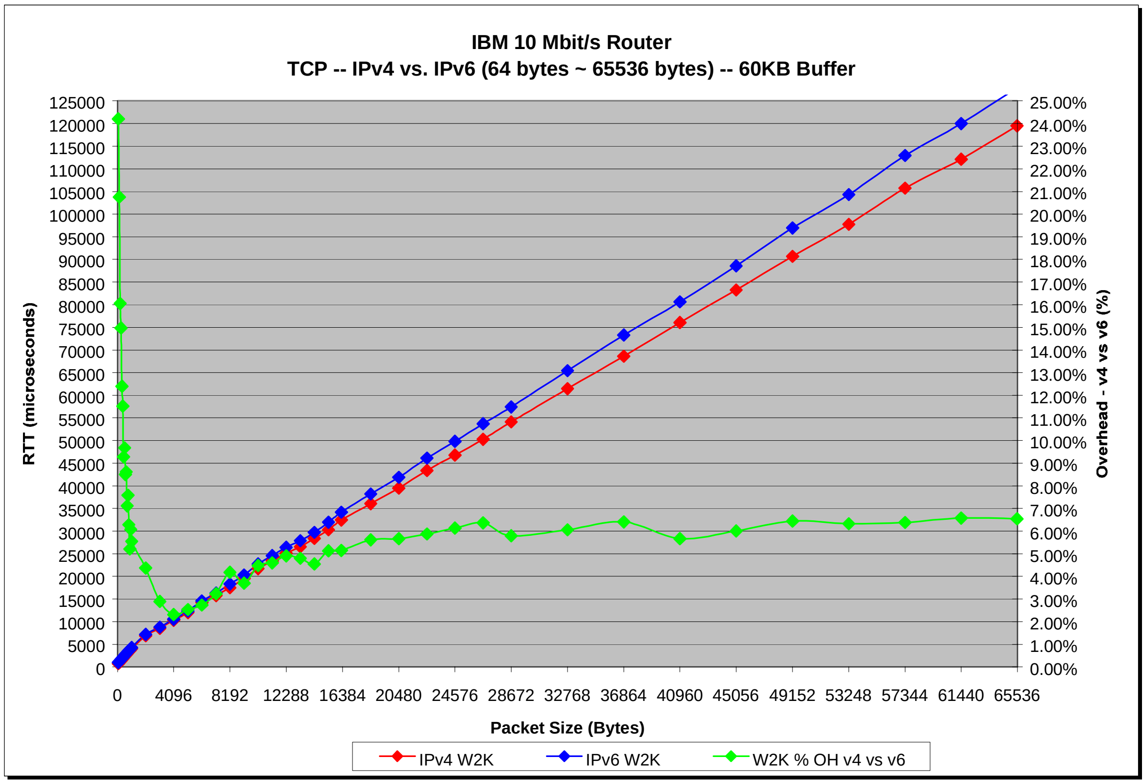


Graph 4. Ericsson Router TCP NT4 and W2K Test# 3

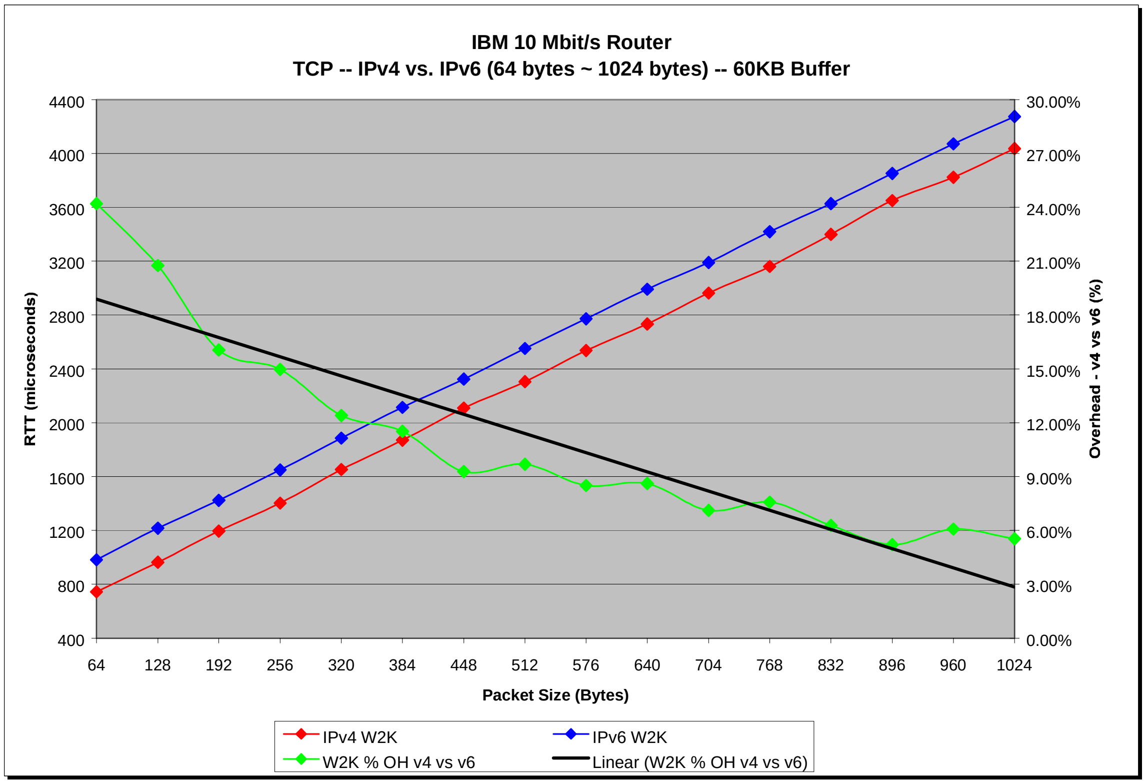
4.1.1.1.2 IBM 10Mbit/s Router

Below, you will find the experimental results performed on the IBM Router with a 10 Mbit/s connection running TCP under Windows 2000.

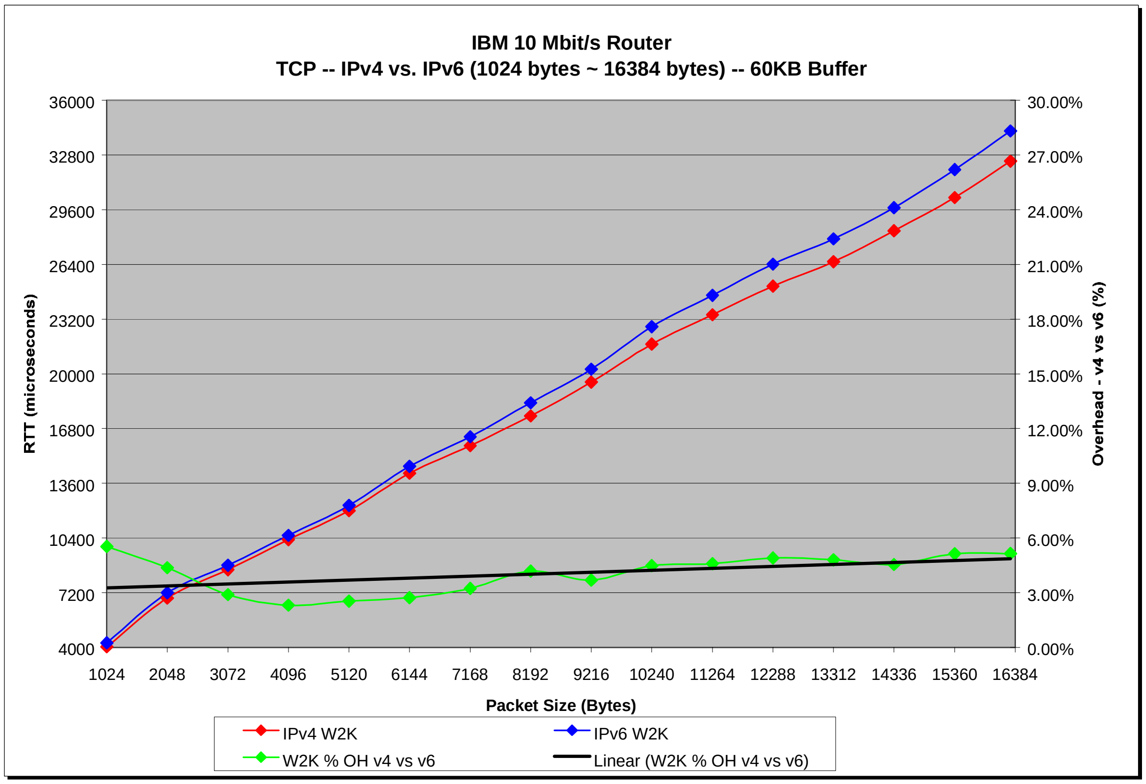
As you will notice in graphs 5 through 8, we have similar results as with the Ericsson router. IPv4 outperforms IPv6 in both Windows NT 4.0 and Windows 2000 anywhere from about 2.5% when using large packets (64KB) to about 25% when using small packets (64B). What seems to be odd is that the overhead seems to very stable between 16KB and 64 KB packets at about 6%, however, the overhead decreases considerably between 4KB and 16KB packets to about 2.5%. It then rises as expected to about 25% overhead, same as before.



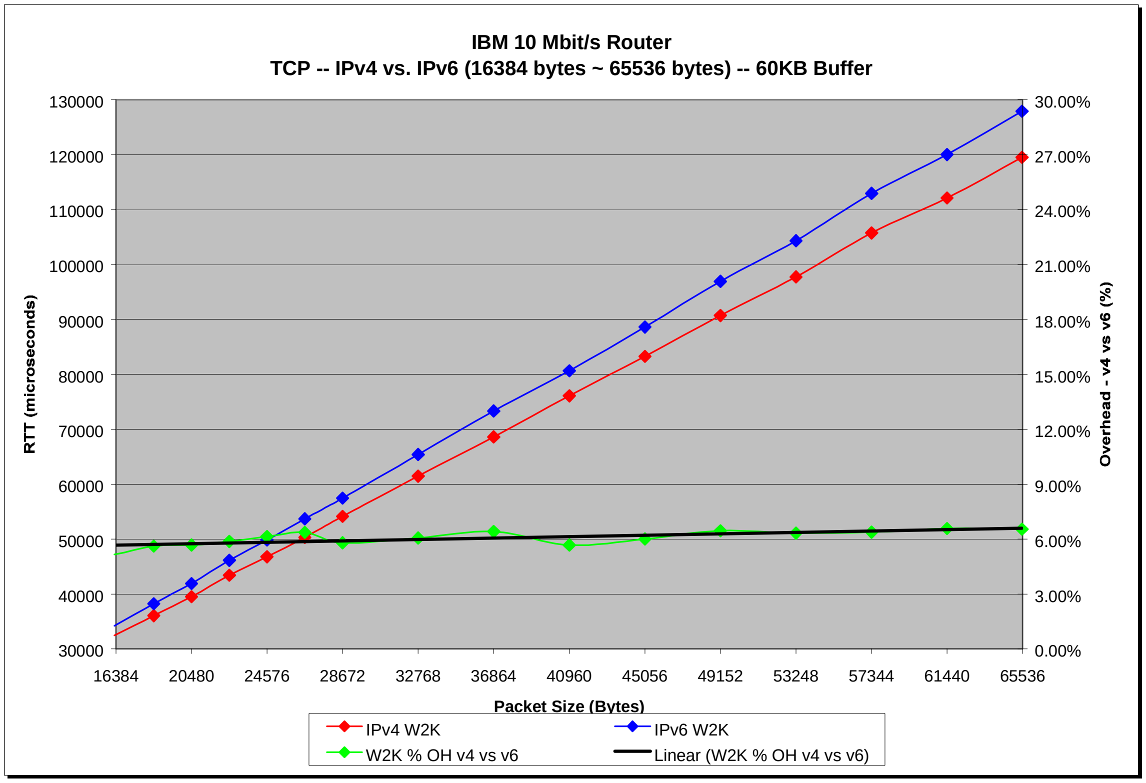
Graph 5. IBM Router TCP W2K Test# 4~6



Graph 6. IBM Router TCP W2K Test# 4



Graph 7. IBM Router TCP W2K Test# 5



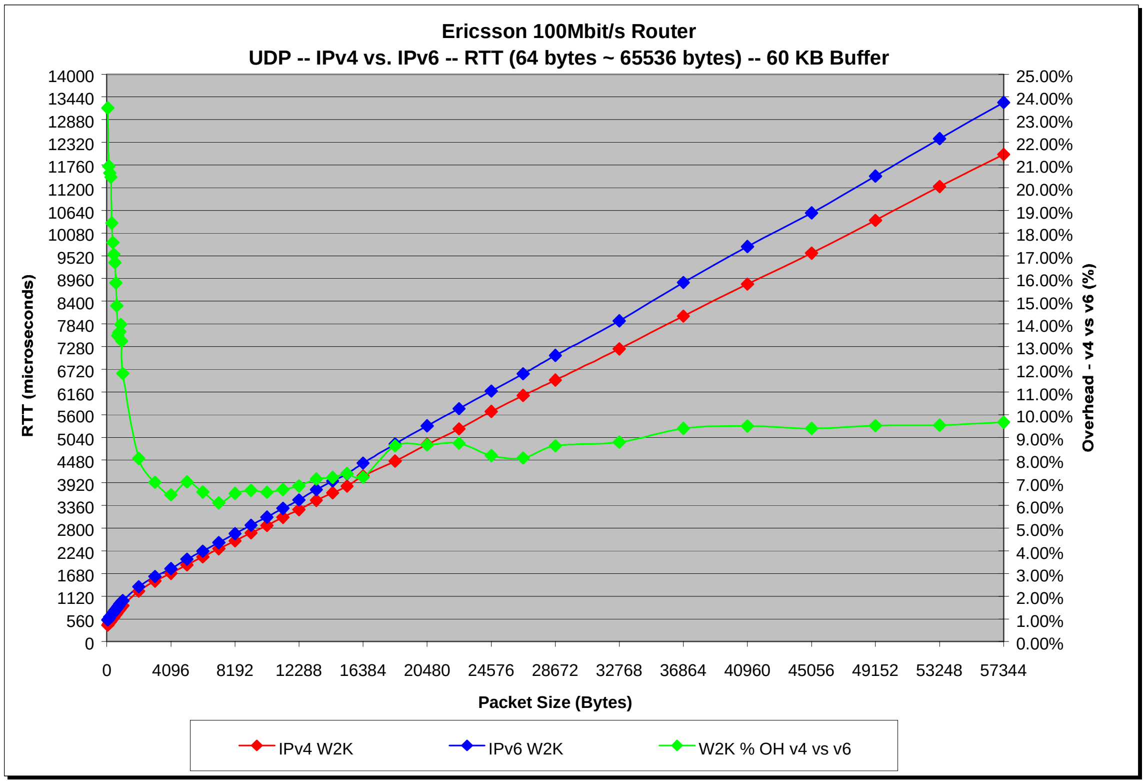
Graph 8. IBM Router TCP W2K Test# 6

4.1.1.2 UDP Protocol

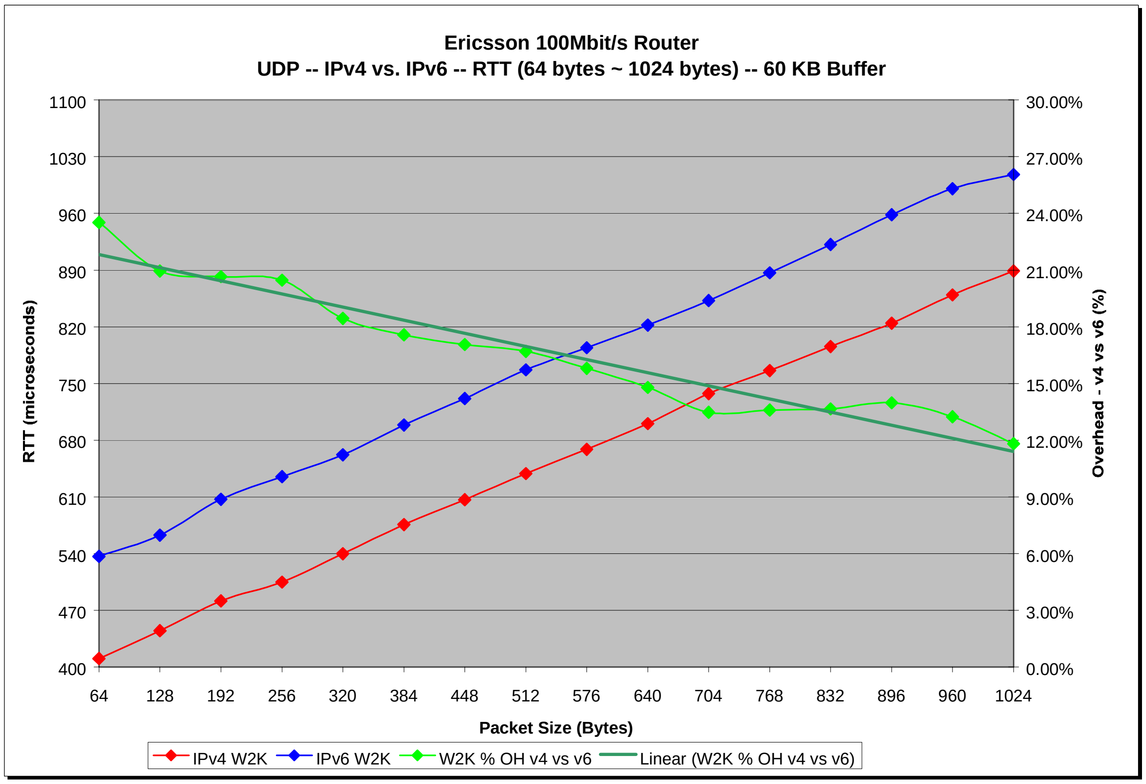
4.1.1.2.1 Ericsson 100Mbit/s Router

Below, you will find the experimental results performed on the Ericsson Router with a 100 Mbit/s connection running UDP under Windows 2000.

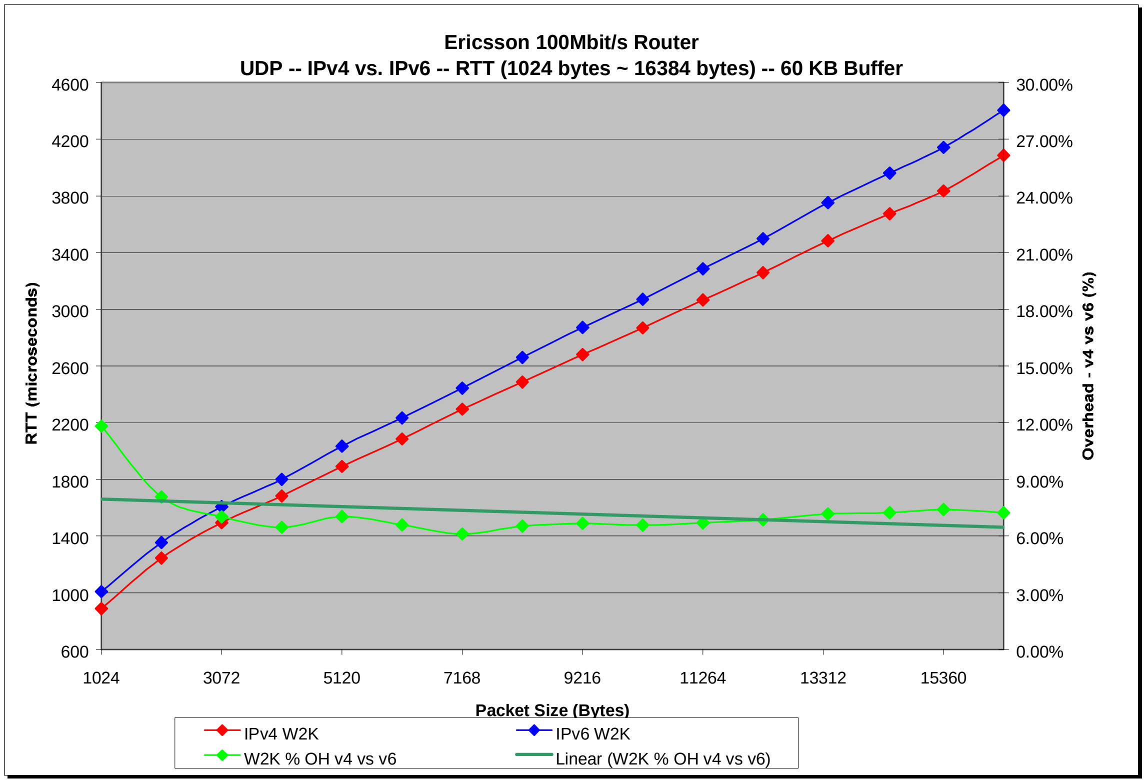
As you will notice in graphs 9 through 12, we have similar results again, however the same odd behavior is occurring as it did in the IBM router using TCP. IPv4 outperforms IPv6 in both Windows NT 4.0 and Windows 2000 anywhere from about 6% when using large packets (64KB) to about 24% when using small packets (64B). What seems to be odd is that the overhead seems to very stable between 16KB and 64 KB packets at about 9%, however, the overhead decreases considerably between 4KB and 16KB packets to about 6%. It then rises as expected to about 24% overhead, same as before.



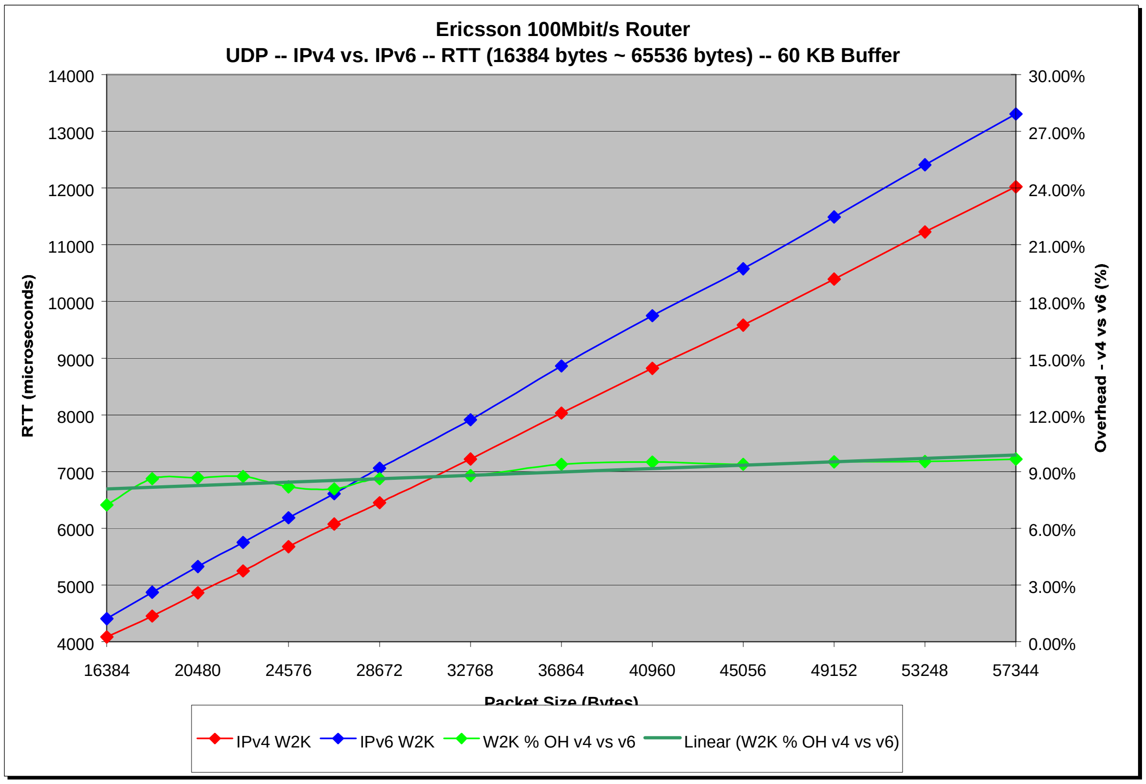
Graph 9. *Ericsson Router UDP W2K Test# 7~9*



Graph 10. *Ericsson Router UDP W2K Test# 7*



Graph 11. Ericsson Router UDP W2K Test# 8

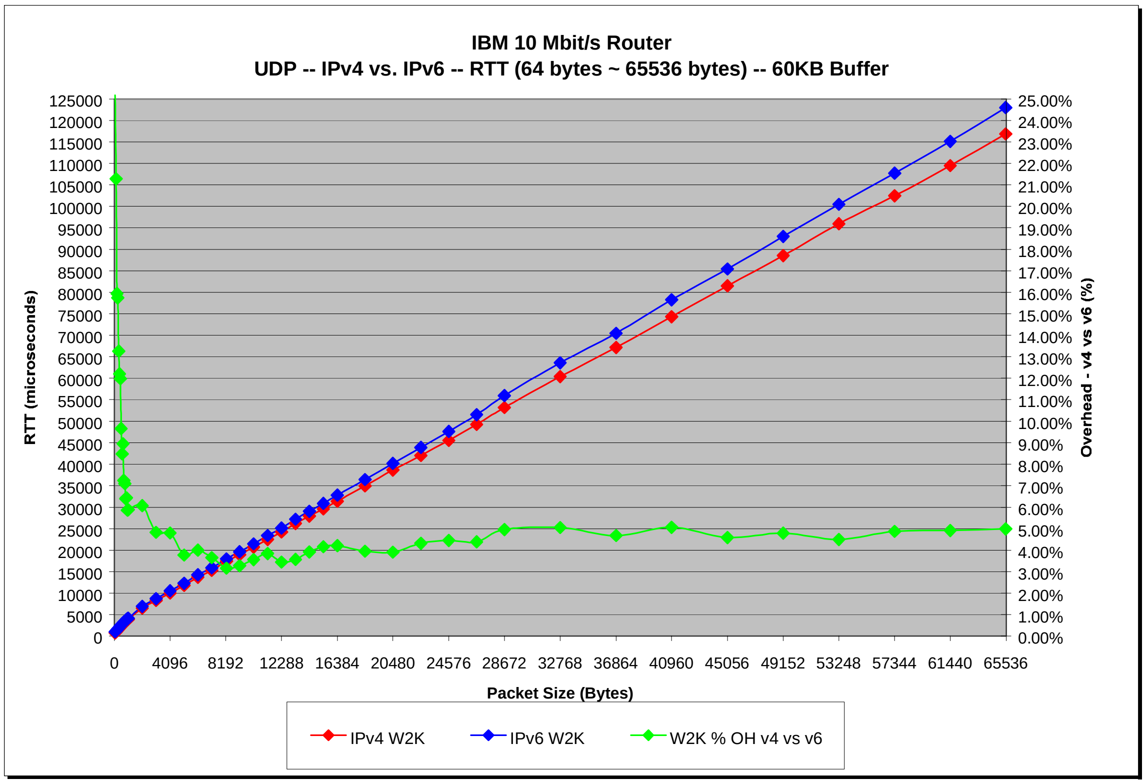


Graph 12. Ericsson Router UDP W2K Test# 9

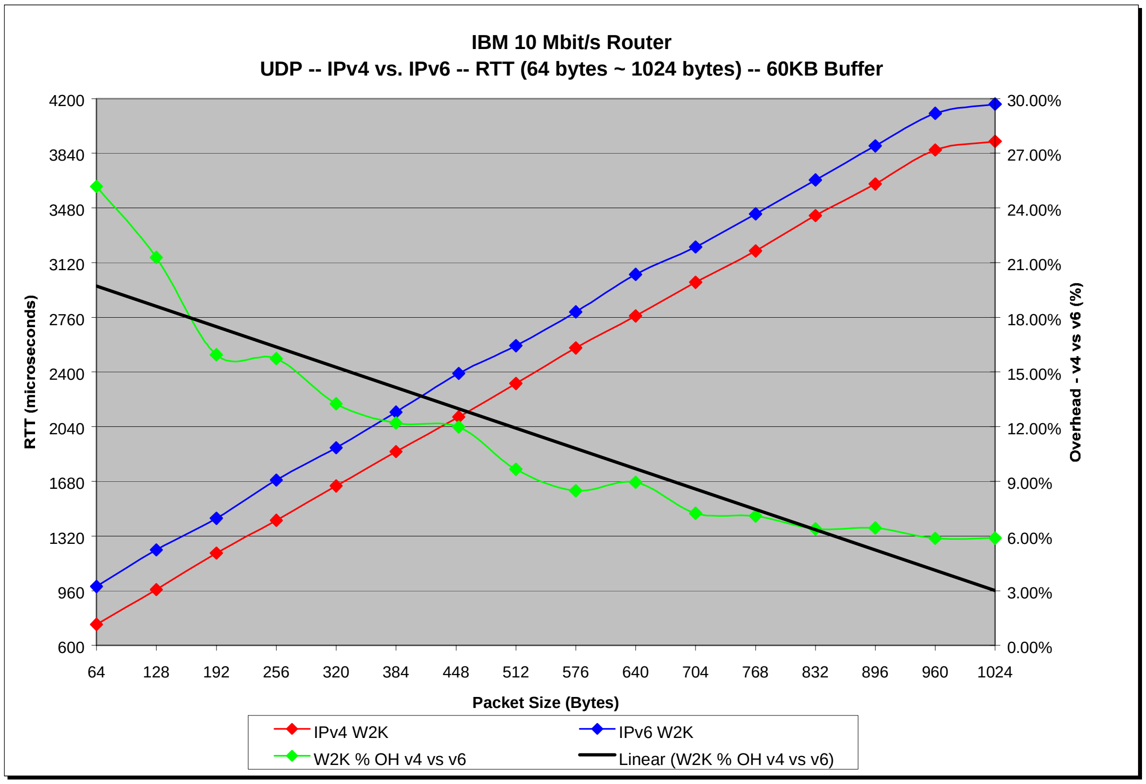
4.1.1.2.2 IBM 10Mbit/s Router

Below, you will find the experimental results performed on the IBM Router with a 10 Mbit/s connection running TCP under Windows 2000.

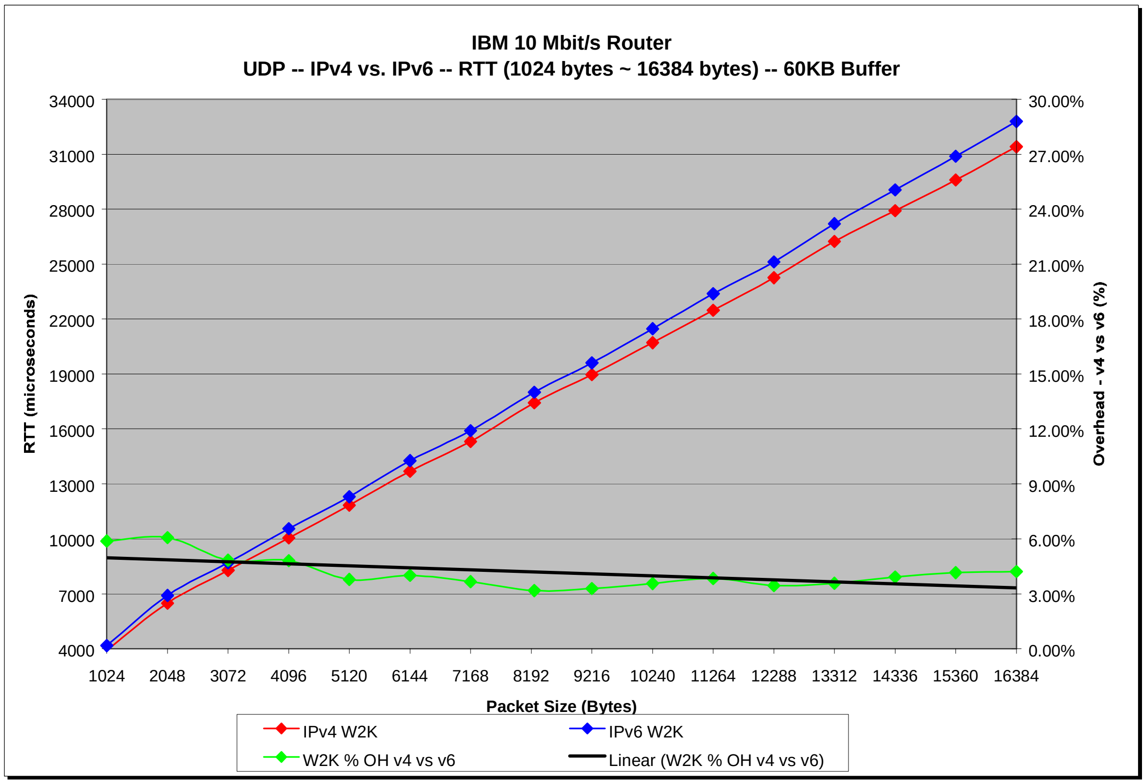
As you will notice in graphs 13 through 16, we have similar results again, however the same odd behavior is occurring as it did in the previous few graphs. IPv4 outperforms IPv6 in both Windows NT 4.0 and Windows 2000 anywhere from about 3% when using large packets (64KB) to about 33% when using small packets (64B). What seems to be odd is that the overhead seems to very stable between 16KB and 64 KB packets at about 5%, however, the overhead decreases somewhat by about 1 ~ 2% until it gets to 8KB packets, after which it increases as expected to 33%.



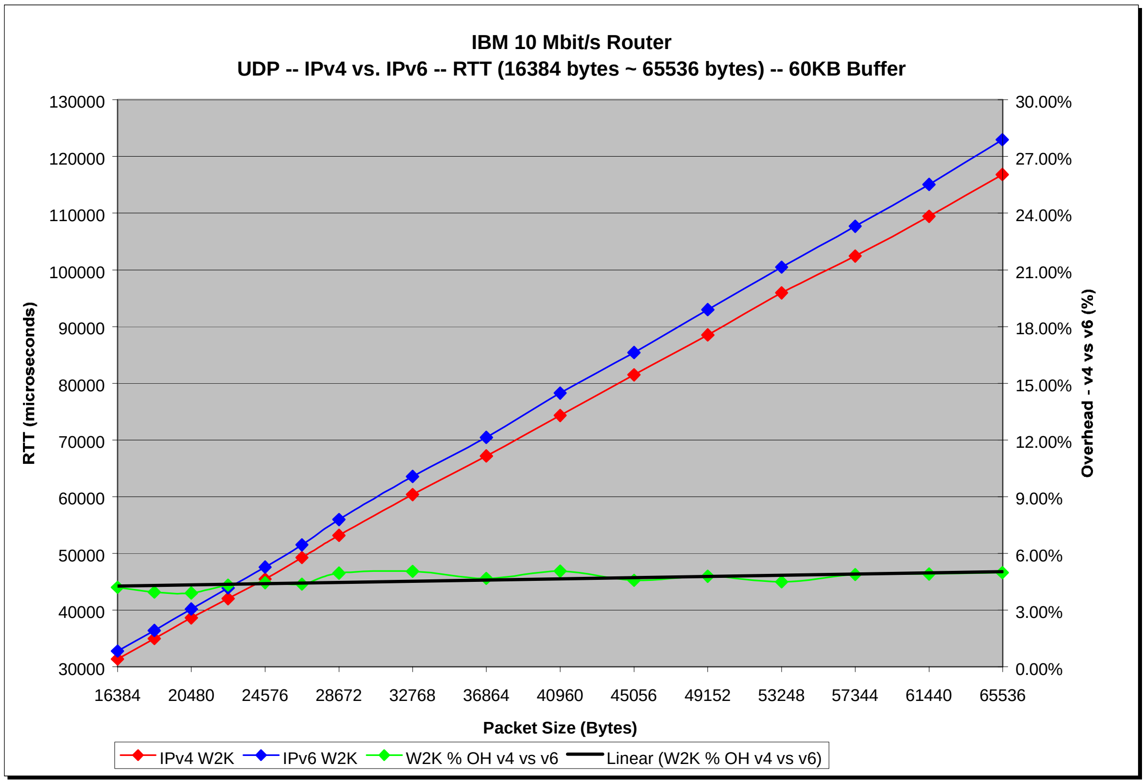
Graph 13. IBM Router UDP W2K Test# 10~12



Graph 14. IBM Router UDP W2K Test# 10



Graph 15. IBM Router UDP W2K Test# 11



Graph 16. IBM Router UDP W2K Test# 12

4.2 Solaris 8.0 OS

4.2.1 Sparc Architecture

4.2.1.1 IPv4 Protocol

4.2.1.1.1 TCP Protocol

4.2.1.1.2 UDP Protocol

4.2.1.2 IPv6 Protocol

4.2.1.2.1 TCP Protocol

4.2.1.2.2 UDP Protocol

4.2.2 Intel Architecture

4.2.2.1 IPv4 Protocol

4.2.2.1.1 TCP Protocol

4.2.2.1.2 UDP Protocol

4.2.2.2 IPv6 Protocol

4.2.2.2.1 TCP Protocol

4.2.2.2.2 UDP Protocol

5.0 Conclusion

According to research findings in [16], IPv6 did not show to have any noticeable overhead using TCP over Ethernet, and had an overhead of 21% using fast Ethernet. They concluded that IPv6 performed so much worse on fast Ethernet due to a CPU bottleneck. According to [18], Microsoft research group itself, claims that there is only a 2.5% performance degradation between IPv6 and IPv4 on Ethernet and only 1.9% performance degradation while running fast Ethernet (I am positive they are using the ideal best case scenario, as their test did not give many details in terms of packet size and buffer size). [18]

My findings concluded that IPv6 incurs a similar overhead regardless of Ethernet or Fast Ethernet. I noticed a 2% to 33% overhead depending what transport protocol I was using and what packet sizes I was sending. The CPU was working about 25% to 50% more on average when processing IPv6 packets when compared to IPv4 packets. This might either be due to added complexity of the IPv6 architecture, or simply due to an inefficient research release IPv6 stack. The huge gap in performance overhead that IPv6 incurs compared to IPv4 might also shrink as the technology matures due to more efficient implementations. [18]

Ideally, IPv6 should only incur a performance degradation of about 1.4% at packets greater than 1500 bytes, and would slowly get to a about 30% for relatively small packets (64 bytes). Once again, I am sure as the technology matures, the protocol will become more efficient and it will behave more as we would expect it to. [18]

6.0 References

[1] William Stallings. *High Speed Networks, TCP/IP and ATM Design Principles.* Pages 444 ~ 457.

[2] Andrew S. Tanenbaum. *Computer Networks, 3rd Edition.* Pages 379 ~ 384.

[8] Goncalves, Marcus and Niles, Kitty. *IPv6 Networks: ICMP, Tunneling, Configuration.*

[9] Zeadally, Sherali. WSU; 2000.

[12] Hinden, Robert M. “IP Next Generation Overview.” May 14, 1995.

[13] Microsoft Corporation. “Introduction to IP Version 6, White Paper”. Pages 1 ~ 38.

[14] Microsoft Corporation. “Microsoft Research IPv6 Release 1.4”. January 13, 2000.

[15] Microsoft Corporation. “Microsoft IPv6 Technology Preview for Windows 2000”. December 12, 2000.

[16] Fiuczynski, Marc E et. Al. “The Design and Implementation of an IPv6/IPv4 Network Adress and Protocol Translator”. 1998.

[17] Karuppiah, Ettikan Kandasamy, et al. “Application Performance Analysis in Transition Mechanism from IPv4 to IPv6”. 2000.

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