Global Information Assurance Certification Paper

Capturing 10G versus 1G Traffic Using Correct Settings!

GCIA Gold Certification
Author: Emilio Valente, evalente@sdsc.edu

Advisor: Stephen Northcutt

Table of Contents

Introduction

ABSTRACT

Proper TCP/IP stack tuning has become an increased factor in host network performance. With 10 and 100 megabit Ethernet, default TCP/IP stack tuning parameters usually were sufficient to utilize the available bandwidth, almost to saturation. However, these settings usually are insufficient for full utilization of Gigabit Ethernet (1 GigE). Since Gigabit technology has become publicly available, there have been numerous attempts to optimize the host TCP/IP stack for it. A set of commonly-known, almost standard tuning parameters have emerged that enable almost complete utilization of the available bandwidth (around 950 Megabits maximum). 10-Gigabits Ethernet (10 GigE) connectivity is a different story. The above common settings don't have the same level of effect -- they don't increase the utilization of the 10 GigE link by the same percentage that they increase the utilization of the 1 Gigabit link. Optimizing the host configuration for this more demanding technology requires a more complex approach.

In this paper, I will describe the steps needed to tune the host TCP/IP stack for optimal throughput for use with 1 GigE network interfaces and 10 GigE network interfaces. I will evaluate throughput by conducting a series of performance tests across several different brands and models of network interface. A second part of this paper will focus on tuning the interfaces and TCP/IP stack to capture network traffic for intrusion detection systems (IDS). Unlike the first part, the focus of tuning for IDS usage will be to limit or avoid packet loss and to increase the maximum speed at which packet loss begins to occur.

I will conclude with recommendations and provide results and reports.

Network Performance

Bandwidth evaluation is an important process that is sometimes overlooked by system designers in distributed as well as non-distributed environments. In the past, the focus was more on network availability than on performance. Currently, since it is very common to see 99.99% of network availability guaranteed, the focus has shifted towards the evaluation and analysis of network applications and performance. The popularity of high-speed connections and faster routers and switches has further driven interest in getting the most performance out of the network.

For internal LANs, network performance using default settings and other configuration parameters can usually meet layer 7 application requirements. Conversely, when long distance connectivity (Internet) is involved, **delays** between sender and receiver can strongly impact bandwidth usage.

The "plug and play" approach to the network configuration does not allow us to benefit from the full network pipe for 3 main reasons:

1. The system TCP/IP stack's buffer sizes (receive and send windows) are too small by default in each OS presently on the market (MS Vista is an exception).

2. Internal LANs have a different structure/technology than WANs in actual physical connections and speed. While we can easily see extremely high network performance across the LAN, performance decreases, sometimes to a fraction of LAN speeds, when moving data across the WAN.

3. The MTU size by default is 1500 bytes. Using an MTU of 9000 bytes (Jumbo Frames) improves network performance by decreasing overhead.

Network TCP/IP Stack Tuning

In 1G connectivity this latency can be greatly reduced by accommodating larger buffer sizes at both directions and at both sites.

In the 1-Gigabit world, TCP/IP buffer tuning has been a straightforward operation and can be considered common to each system based on its OS. A set of commonly-known, almost standard tuning parameters have emerged and a comprehensive tutorial and explanation of the settings that should be changed in the kernel (Linux) and registry (Windows) to improve network performance can be found at the PSC website.[1] I have used that reference as a guideline and applied custom values to those settings for better results - almost saturating the link with over 950 Megabits per second in the LAN - during my experiments. The same settings have been applied to the long-distance tests (Chicago and New York from San Diego) through Abilene Network.[2]

10-Gigabit network tuning is completely different. Like 1-Gigabit, tuning the kernel is involved. However, 10Gigabit places significantly more demand on the host system when compared to 1-Gigabit. Most hosts can move data around internally at 125 megabytes per second, roughly the speed necessary to keep a 1-Gigabit link fully utilized. On the other hand, 10-Gigabit requires the host to move data around at over 1.25 gigabytes per second to keep the link fully utilized. Thus, the host hardware (CPU, bus speed, memory speed, and interconnects) becomes an important factor in 10-Gigabit performance.

To complicate matters, the network interface tuning parameters and values differ, from brand to brand and model to model, because NIC manufacturers adopt different ways to handle buffers, sockets, interrupts, and memory caches. Several network and driver parameters were modified to obtain maximum utilization of the network pipe. These settings have demonstrated a great impact

on the jumbo frame enabled tests, as subsequently explained.

Testing Methodology

The following will explain how the end-to-end methodology will be used to execute network performance testing using "in production" equipment. The goal is to achieve the highest network throughput (wire-speed) of 1 Gigabit Ethernet (GigE) and 10 GigE networks.

This methodology quantifies how fast data travel from point A to point B and establishes the "bandwidth headroom", the difference between the nominal maximum wire-speed and what applications/tools actually achieve. Bandwidth headroom is a significant measurement since a network with a large amount of headroom is better equipped to handle sudden peaks of traffic; however, reserving headroom can also mean reducing the potential availability or performance of applications on the network. Thus, a balance must be achieved where there is enough headroom to accommodate additional requests, but not too much to limit existing applications. By tuning the hosts and interfaces to increase the nominal maximum wire speed, the bandwidth headroom can be increased "for free."

Iperf[3] has been used as the networking testing tool. Iperf can measure the maximum bandwidth, permits the tuning of a range of parameters like window size, mtu size and numbers of bites to transmit. Iperf reports bandwidth, latency, jitter, packet loss; and, if the system has multiple NICs, the single test can be bound to a specific interface, avoiding false results.

TESTING NETWORK TOPOLOGY

10GigE and 1 GigE Testing LAN Topology

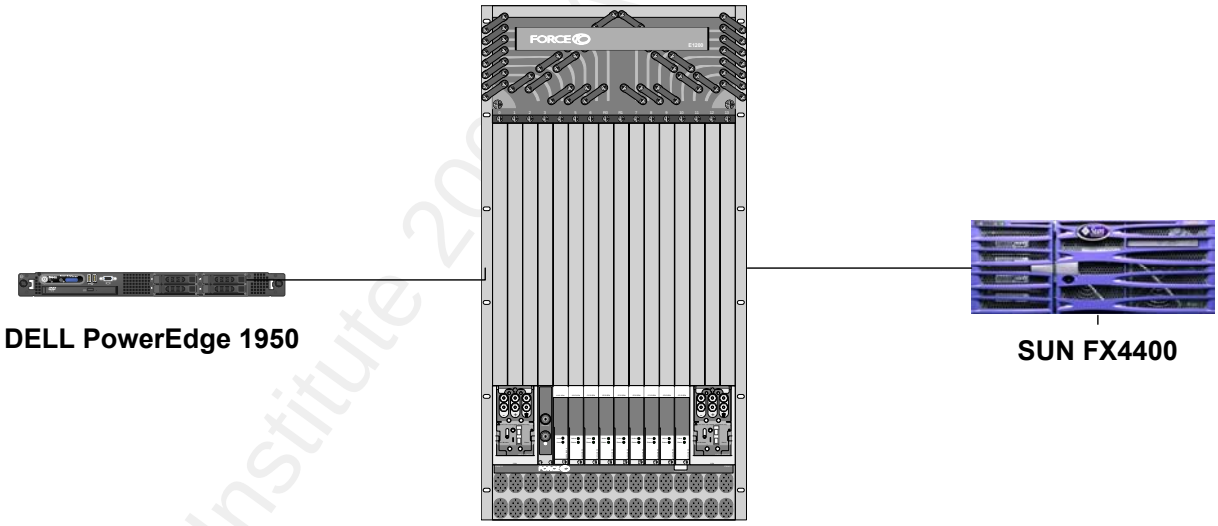


Table 1

10GigE Testing WAN Topology

Internet2 (Abilene) network

```
[evalente@server]# traceroute SAN DIEGO >>> WASHINGTON
traceroute to **.***.**.**, 30 hops max, 40 byte packets
 1  **********.sdsc.edu (**.**.**.**)  0.612 ms  0.563 ms  0.554 ms
 2  ********.sdsc.edu (**.**.**.**)  0.759 ms  0.751 ms  0.951 ms
 3  ********.sdsc.edu (**.**.**.**)  0.724 ms  0.717 ms  0.923 ms
 4  *********sdsc-10ge.cenic.net (**.**.**.**)  5.933 ms * *
 5  *****lax-*****.10ge.cenic.net (**.**.**.**)  4.362 ms  4.353 ms  4.332 ms
 6  *****.hous.internet2.edu (**.**.**.**)  36.178 ms  36.201 ms  36.396 ms
 7  *******.internet2.edu (**.**.**.**)  59.750 ms  59.735 ms  59.720 ms
 8  *******.wash.internet2.edu (**.**.**.**)  73.234 ms  73.444 ms  73.211 ms
 9  *******.wash.internet2.edu (**.**.**.**)  73.195 ms  73.182 ms  73.171 ms
```

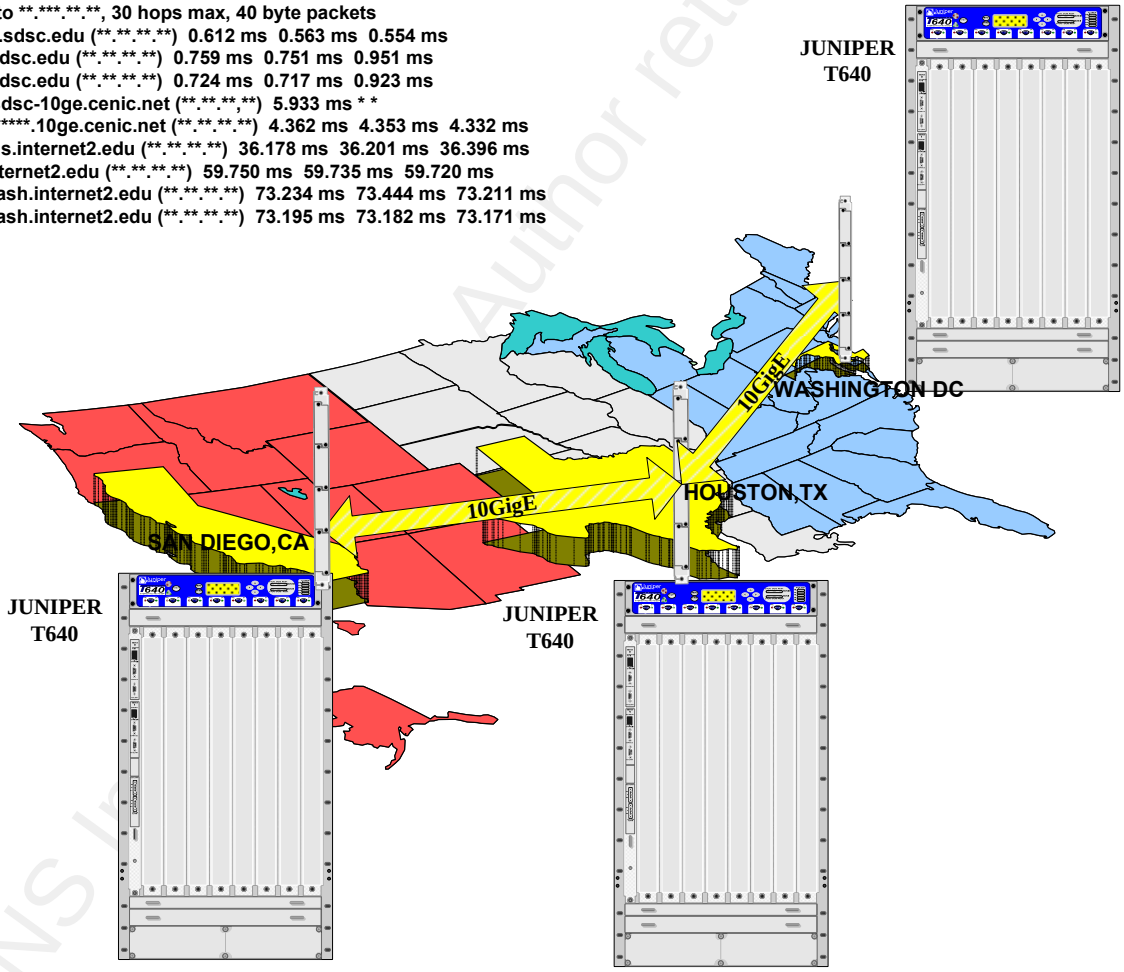


Table 2

10GigE and 1 GigE Capture Testing LAN Topology

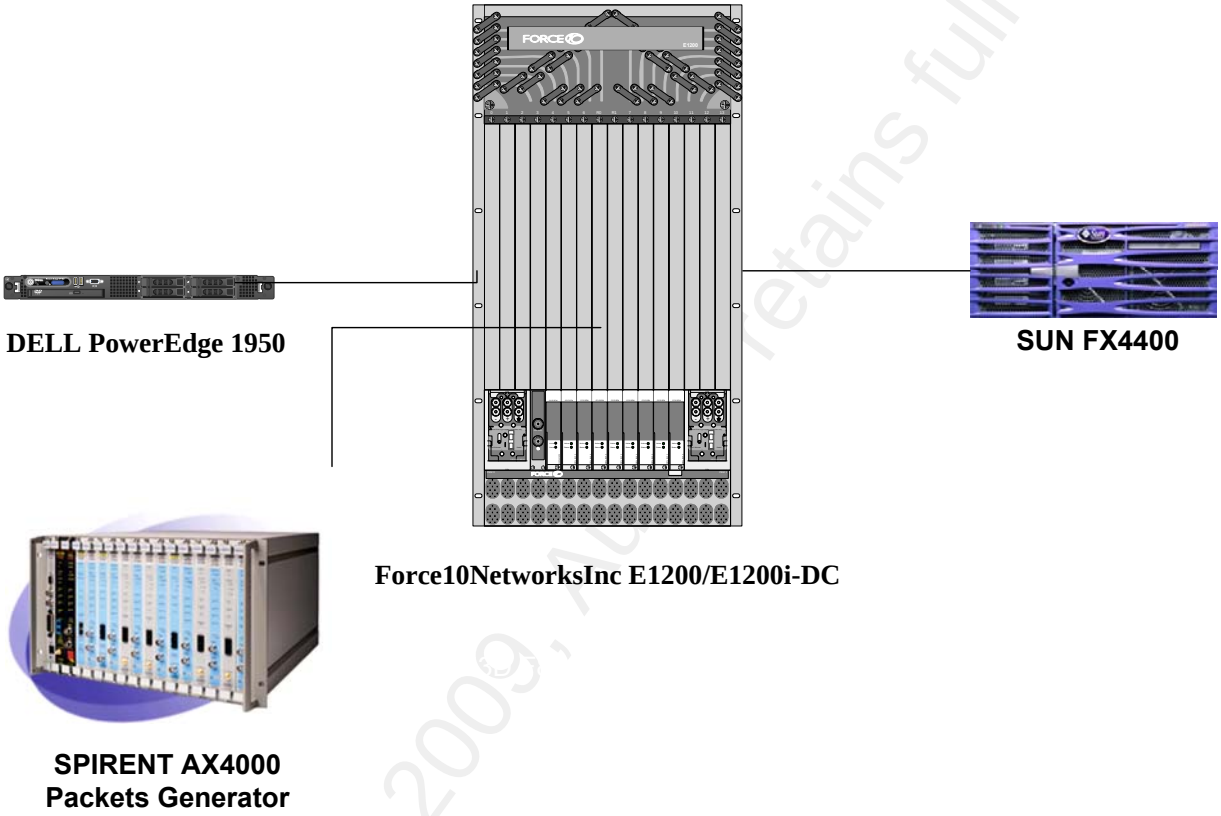


Table 3

Step-by-Step Tuning: 1 Gigabit

The following are the recommended settings for the 1 GigE tuning:

```
echo 1 > /proc/sys/net/ipv4/tcp_moderate_rcvbuf
echo 33554432 > /proc/sys/net/core/wmem_max
echo 33554432 > /proc/sys/net/core/rmem_max
echo "4096 33554432 33554432" > /proc/sys/net/ipv4/tcp_rmem
echo "4096 33554432 33554432" > /proc/sys/net/ipv4/tcp_wmem
```

Edit the sysctl.conf file locate in /etc folder, adding the above echo commands. This will avoid the problem, which could occur at the next reboot, of the system reinitializing these settings to the default OS values.

End-to-End 1 Gigabit Network Tests:

- Local Host (based on motherboard, cpu, bus and memory) - this is the fastest the system can sustain in speed)

OS: Linux Kernel 2.6 NIC: Embedded NC324i MTU:

```
[emilio@server]$ iperf -s 127.0.0.1 -p 5555 -w 30MB &
Server listening on TCP port 5555
TCP window size: 60.0 MBytes (WARNING: requested 30.0 MBytes)
[emilio@server]$ iperf -c 127.0.0.1 -w 30MB -p 5555
[ 4] local 127.0.0.1 port 5555 connected with 127.0.0.1 port 34500
------------------------------------------------------------
Client connecting to 127.0.0.1, TCP port 5555
TCP window size: 60.0 MBytes (WARNING: requested 30.0 MBytes)
------------------------------------------------------------
[ 3] local 127.0.0.1 port 34500 connected with 127.0.0.1 port 5555
[ 3]  0.0-10.0 sec 8.19 GBytes 7.03 Gbits/sec
[ 3] MSS size 16384 bytes (MTU 16424 bytes, Ethernet)
```

Table 4

Explanation of Table 1: The iperf server is listening on port TCP 5555, the iperf requested windows size for the test equals to 60 Mbytes (iperf doubled the request of 30 MB). The client

(localhost) is connected to the same port 5555, the test lasts 10 seconds (0.0 - 10.0), the amount of packets moved is 8.19 GBytes in size and their transfer speed (internally) is **7.03 GBits** per second. The MTU id 16424 since the loopback default MTU is 16436.

- Back-to-Back NO tuning TCP/IP stack

OS: Linux Kernel 2.6 NIC: PCI Sysconnect D9321 MTU: 1500

Back-to-Back testing involves the connection from one server to another without the intervention of the switch.

```
[evalente@server]# ./iperf -c 10.2.2.2 -B 10.1.1.1
------------------------------------------------------------
Client connecting to 10.2.2.2, TCP port 5001
Binding to local address 10.1.1.1
TCP window size: 16.0 Kbytes (default)
------------------------------------------------------------
[ 5] local 10.1.1.1 port 5001 connected with 10.2.2.2 port 5001
[ 5] 0.0- 10.0 sec 1.12 GBytes 943 Mbits/sec
[ 5] MSS size 1448 bytes (MTU 1500 bytes, Ethernet)
```

Table 5

Explanation of Table 2: The iperf server 10.2.2.2 is listening on TCP port 5001, the windows size is 16.0 KByte the default, since the receive buffer by default on Linux are 16384 bytes and the client 10.1.1.1 is connected to the same port 5001, the test lasts 10 seconds (0.0 - 10.0), the amount of packets moved is 1.12 GBytes in size and their transfer speed is **943 Mbits/sec**.

- **Back-to-Back WITH tuning TCP/IP stack**

OS: Linux Kernel 2.6 NIC: Embedded NC324i MTU: 9000

```
[evalente@server]# ./iperf -c 10.1.1.3 -B 10.1.1.1 -p 5008
------------------------------------------------------------
Client connecting to 10.1.1.3, TCP port 5001
Binding to local address 10.1.1.1 TCP window size: 33554432 Bytes
(default)
------------------------------------------------------------
[7] local 10.1.1.1 port 5001 connected with 10.1.1.3 port 5001
[7] 0.0-10.0 sec 1.263 GBytes 1.007 Gbits/s.
[7] MSS size 8948 bytes (MTU 8988 bytes, Ethernet)
```

Table 6

Explanation of Table 3: The iperf server 10.1.1.3 is listening on TCP port 5001, the windows size is 33554432 Bytes, the receive default buffer after tuning, the client 10.1.1.1 is connected to the same port 5001, the test lasts 10 seconds (0.0 – 10.4), the amount of packets moved is 1.263 GBytes in size and their transfer speed is **1.007 Gbits/s**. This means that the back-to-back test is saturating the link at wire speed.

- Switch Fabric (Cisco catalyst 3760, 6509) NO tuning TCP/IP stack and insignificant latency (0.123 microseconds)

OS: Linux Kernel 2.6 NIC: Embedded NC324i MTU: 9000

```
[evalente@server]# iperf -c 10.1.1.3 -B 10.1.1.1 -p 5002
------------------------------------------------------------
Client connecting to 10.1.1.3, TCP port 5002
Binding to local address 10.1.1.1
TCP window size: 16.0 KBytes (default)
------------------------------------------------------------
[3] local 10.1.1.1 port 5002 connected with 10.1.1.3 port 5002
[3] 0.0-10.0 sec 1.10 GBytes 932 Mbits/sec
[3] MSS size 8948 bytes (MTU 8988 bytes, Ethernet)
```

Table 7

Explanation of Table 4: The iperf server 10.1.1.3 is listening on TCP port 5002, the windows size is 16.0 KByte the default, since the receive buffers by default on Linux are 16384 bytes, the client 10.1.1.1 is connected to the same port 5002, the test last 10 seconds (0.0 – 10.4) the amount of packets moved is 1.10 GBytes in size and their transfer speed is **932 Mbits/sec**.

- Switch Fabric (Cisco Catalyst 3760, 6509) WITH tuning TCP/IP stack

OS: Linux Kernel 2.6 NIC: Embedded NC324i MTU: 9000

```
[evalente@server]# iperf -c 10.1.1.3 -B 10.1.1.1 -p 5001
------------------------------------------------------------
Client connecting to 10.1.1.3, TCP port 5001
Binding to local address 10.1.1.1
```

```
TCP window size: 33554432 Bytes (default)
------------------------------------------------------------
[5] local 10.1.1.1 port 5001 connected with 10.1.1.3 port 5001
[5] 0.0-10.0 sec 1.12 GBytes 979 Mbits/sec
[5] MSS size 8948 bytes (MTU 8988 bytes, Ethernet)
```

Table 8

Explanation of Table 5: The iperf server 10.1.1.3 is listening on TCP port 5001, the windows size is 60.0 MBytes, iperf doubled the request of 30 Mbytes, the client 10.1.1.1 is connected to the same port 5001, the test lasts 10 seconds (0.0 - 10.4), the amount of packets moved is 1.12 GBytes in size and their transfer speed is **979 Mbits/sec**. This means that without delay, since it is still running in the LAN, the test is almost using the total amount of the bandwidth available.

- Long-distance Router (Juniper T320) NO tuning TCP/IP stack

OS: Linux Kernel 2.6 NIC: PCI Sysconnect D9321 MTU: 9000

```
[evalente@server]# iperf -c xxx.xxx.xxx.xx -p 5001
------------------------------------------------------------
Server listening on TCP port 5001
Binding to local address xx.xx.xx.xx (Chicago)
TCP window size: 87380 Byte (default)
------------------------------------------------------------
[14] local xx.xx.xx.xx port 5001 connected with xx.xx.xx.xx port
5001
[14] 0.0-10.2 sec 720655667 Bytes 584433456 bits/sec
[14] MSS size 8948 bytes (MTU 8988 bytes)
------------------------------------------------------------
[evalente@server]# iperf -c xxx.xxx.xxx.xx -p 5002
------------------------------------------------------------
Server listening on TCP port 5002
Binding to local address xx.xx.xx.xx (New York)
TCP window size: 87380 Byte (default)
------------------------------------------------------------
[15] local xx.xx.xx.xx port 5002 connected with xx.xx.xx.xx port
5002
[15] 0.0-10.1 sec 643869824 Bytes 535455566 bits/sec
[15] MSS size 8948 bytes (MTU 8988 bytes)
------------------------------------------------------------
```

Table 9

```
[evalente@server]# iperf -c xxx.xxx.xxx.xx -p 5001
------------------------------------------------------------
Client connecting to xx.xx.xx.xx TCP port 5001
Binding to local address xx.xx.xx.xx (Chicago)
TCP window size: 33554432 Byte (default)
------------------------------------------------------------
[ 7] local xx.xx.xx.xx port 5001 connected with xx.xx.xx.xx port
5001
[ 7] 0.0-10.0 sec 1030627328 Bytes 822837427 bits/sec
[ 7] MSS size 8948 bytes (MTU 8988 bytes, unknown interface)
------------------------------------------------------------
[evalente@server]# iperf -c xxx.xxx.xxx.xx -p 5002
Client connecting to xx.xx.xx.xx, TCP port 5002
Binding to local address xx.xx.xx.xx (New York)
TCP window size: 33554432 Byte (default)
------------------------------------------------------------
[ 7] local xx.xx.xx.xx port 5001 connected with xx.xx.xx.xx port
5001
[ 7] 0.0-10.0 sec 993869824 Bytes 793544638 bits/sec
[ 7] MSS size 8948 bytes (MTU 8988 bytes, unknown interface)
```

Table 10

Explanation of Table 6 and 7: These two long-distance tests have demonstrated that buffer size has a great influence on increasing network performances where delay is involved. In the first two tests, default values are only sufficient to achieve around 50% (584 and 535 MB/s) of the total available bandwidth. In the second tests, I have increased the TCP/IP stack using a value equals to 33554432 for the buffer settings (receiving and sending) and I was able to use between 80 and 82% of the available bandwidth across the country.

Step-by-Step Tuning: 10 Gigabit Ethernet

As previously mentioned, an important factor that was not very relevant in the 1GigE testing, is the amount of resources available by the system. CPU power, RAM memory, bus speed interface - all play a significant role in the performances of 10GigE tests.

Two 10GigE cards PCI Express have been installed and tested in

two different system architectures.

The following are the recommended settings that have demonstrated the best performance.

(PCI Express is a more modern generation of 10GigE card after the old PCI X).

SYSTEMS:

1. DELL - PowerEdge Energy Smart 1950 III CPUs: 2 Quad-Core (8 CPUs Intel® Xeon® X5355, 2x4MB Cache, 2.66GHz, 1333MHz FSB RAM: 16GB 667MHz (4x4GB), Dual Ranked DIMMs, Energy Smart
2. SUN Microsystems - Fire X4440 CPUs: 4 Quad-Core (16 CPUs AMD Opteron processors 8347) RAM: 16 GB.

10 GigE CARDS:

- Sun Microsystems - DUAL 10 GigE XFP Express
- Myricom - MYRI-10GE – Express.

OS:

On both systems is Red Hat Linux AS 5 - 64 bits Linux localhost.localdomain 2.6.18-92.el5 #1 SMP Tue Apr 29 13:16:15 EDT 2008 x86_64 GNU/Linux.

CPU/Core architecture:

In the DELL PowerEdge 1950 each of the quad-core Intel® Xeon® X5355 CPUs have two L2 cache units. Each L2 cache unit (4096 KBytes) is shared between two cores.

In this architecture, core 0-2-4-6 are on CPU #0 and 1-3-5-7 are

on CPU #1.

In the SUN Fire X4440 each core of the AMD Opteron processors 8347 has 512 KBytes of L2 cache not shared. In this architecture, cores #0-1-2-3 are on CPU #1, 4-5-6-7 on CPU #2, 8-9-10-11 on CPU #3, 12-13-14-15 on CPU#4.

****NOTICE****

Default values, in my tuning, refer to the latest (at the time I am writing) Linux Red Hat AS 5 kernel 2.6.18-92 settings, the one I have tested.

In all my examples below, substitute "eth4" with your local configured interface connected to the 10GigE NIC.

In **bold** are the commands. Also, the following are the suggested settings and explanations (in *Italic*) by the card manufacturer that:

1) are the most beneficial (among the many suggested)
2) I have tested
3) I have incorporated in the tuning procedure.

Remember that CPUs contain Cores. You will notice that some outputs below refer to "cpu" or "processor", even though it actually means "core". This is because machine terminology precedes the invention of the "core".

Let's begin!

SUN DUAL 10GigE Fiber XFP Express 8x[4]:

a) Check the driver version and make sure you have the latest from SUN Microsystems:

```
[evalente@server]# ethtool -i eth4
driver: nxge
version: 2.2.1
firmware-version: 2XGF PXE1.48 FCode 3.12
bus-info: 0000:81:00.1
```

b) Customize memory settings buffers for TCP/IP and Core tuning:

- Memory settings buffers suggested by SUN[4] for TCP/IP tuning:

```
### IPV4 specific settings
# turns TCP timestamp support off, default 1, reduces CPU use
net.ipv4.tcp_timestamps = 0
```

```
# turn SACK support off, default on systems with a VERY fast bus ->
# memory interface this is the big gainer
net.ipv4.tcp_sack = 0
# sets min/default/max TCP read buffer, default 4096 87380 174760
net.ipv4.tcp_rmem = 10000000 10000000 10000000
# sets min/pressure/max TCP write buffer, default 4096 16384 131072
net.ipv4.tcp_wmem = 10000000 10000000 10000000
# sets min/pressure/max TCP buffer space, default 31744 32256 32768
net.ipv4.tcp_mem = 10000000 10000000 10000000
```

- Memory settings buffers suggested by SUN[4] for Core tuning:

```
### CORE settings
# maximum receive socket buffer size, default 131071
net.core.rmem_max = 524287
# maximum send socket buffer size, default 131071
net.core.wmem_max = 524287
# default receive socket buffer size, default 65535
net.core.rmem_default = 524287
# default send socket buffer size, default 65535
net.core.wmem_default = 524287
# maximum amount of option memory buffers, default 10240
net.core.optmem_max = 524287
# number of unprocessed input packets before kernel starts dropping
# them, default 300
net.core.netdev_max_backlog = 300000 (default 1000)
```

- Memory settings buffers for long distance (30 to 70 ms delay) and large data transfer (my tested and suggested settings):

TCP/IP memory tuning

```
net.ipv4.tcp_timestamps = 0
net.ipv4.tcp_sack = 0
net.ipv4.tcp_rmem = 4096 134217728 134217728
net.ipv4.tcp_wmem = 4096 134217728 134217728
```

Core memory tuning

```
net.core.rmem_max = 134217728
net.core.wmem_max = 134217728
net.core.netdev_max_backlog = 250000
```

To modify the above settings, add the commands to the **sysctl.conf**

file, located in /etc folder and then run:

/sbin/sysctl -p

By following this procedure, it will avoid the problem that, at next reboot, your system will reinitialize these settings to the default OS values. (I prefer this to the "echo" commands described earlier for the 1 GigE tuning).

c) Interrupt Coalescing

"rx-usecs and rx-frames control the RX interrupt rate per RX DMA channel. RX interrupt will be generated after rx-frames have been received or after rx-usecs time interval if fewer than rx-frames have been received within the interval. For low latency applications, set rx-usecs to a smaller value. For bulk traffic, use larger values of rx-usecs and control the rate with rx-frames. For low latency applications, set rx-usecs to a smaller value. For bulk traffic, use larger values of rx-usecs."[4]

Check the interrupt coalescing values (default value for rx-usecs is 8 for rx-frames is 512) with:

ethtool -c eth4

For the SUN card we have two different settings based on the architecture:

Since the SUN Fire X4440 has the AMD Opteron that supports DMA (Direct Memory Access), the best value for rx-usecs is 1000, and rx-frames is 1.

For the Dell 1950, since the Intel CPU (on board) has a shared L2 cache, the values of rx-usecs of 0 and rx-frames of 512 (default) are the one I recommend.

Set the value of rx-usecs to 0 (or 1000) with:

ethtool -C eth4 rx-usecs 0 (or 1000)

Set the value of rx-frames to 1 (or 512) with:

ethtool -C eth4 rx-frames 1 (or 512)

d) Check and eventually turn TCP segmentation OFF

By turning off tcp segmentation offload, it will let the kernel

process packets.

To display this category of settings, type the following command:

```
[evalente@server007 ipv4]# /sbin/ethtool -k eth4
Offload parameters for eth4:
rx-checksumming: on
tx-checksumming: on
scatter-gather: on
tcp segmentation offload: on
udp fragmentation offload: off
generic segmentation offload: off
To turn TSO off type the following command:
[evalente@server007 ipv4]# /sbin/ethtool -K eth4 tso off
```

Now TSO is off as listed below:

```
[evalente@server007 ipv4]# /sbin/ethtool -k eth4
Offload parameters for eth4:
rx-checksumming: on
tx-checksumming: on
scatter-gather: on
tcp segmentation offload: off
udp fragmentation offload: off
generic segmentation offload: off
```

e) TXQUEUELEN

"For WAN transfers, it was discovered that a setting of 2,000 for the txqueuelen is sufficient to prevent any send stalls from occurring."[5]

Default value for txqueuelen is 1000. I have successfully tested a value of 2500. Type the follow command to change it:

```
/sbin/ifconfig eth4 txqueuelen 2500
/sbin/ifconfig eth4:
eth4 Link encap:Ethernet HWaddr 00:14:AD:56:43:8A
 inet addr:**.**.**.**5 Bcast:**.**.*85.255 Mask:255.255.255.0
 inet6 addr: fg40::260:deff:ff46:f46a/64 Scope:Link
 UP BROADCAST RUNNING MULTICAST MTU:9000 Metric:1
 RX packets:239279000 errors:0 dropped:0 overruns:0 frame:0
```

```
TX packets:127273234 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:2500
RX bytes:1565907292058 (1.4 TiB) TX bytes:752878505130 (701.1 GiB)
Interrupt:194
```

f) CPU Binding/Affinity (or better: Core Binding/Affinity)

My goal with this task is to identify the best combination of CPU/core usage for the best test performances (in this case) and ultimately give the best suggestions to everyone for improving large data transfers.

- The following are the steps for CPU binding:

1) If it is running, IRQBALANCE service must be killed

This allows one to obtain control of how interrupts are handled in the CPU (Core, L2cache sharing, internal bus).

2) Direct to only one CPU/core tx and rx data:

The SUN DUAL 10GigE Fiber XFP Express 8x has 20 channels. Channels identified by number 8 through number 15 (8 channels) are RX (receive channels) while channels identified by number 12 through 23 (12 channels) are TX (transmit channel). They can be displayed by typing:

```
[root@dell1950 evalente]# ethtool -S eth4
```

Each channel can trigger one or more cores in the CPU that can be randomly picked by the driver. To narrow this selection to one specific core, use the nxge.conf file that comes with the Sun card driver.

To add one rule, proceed as follows:

```
[root@dell1950 bin]# nxge_config eth4 show_rule
Flow Rule entries:
==================
[root@dell1950 bin]# nxge_config | grep add_rule
 add_rule <tcp | udp | sctp | usr <user_proto>>
root@dell1950 bin]# nxge_config eth4 add_rule tcp dport 5001 chan 0
Rule added with ID [153]
```

```
[root@dell1950 bin]# nxge_config eth4 show_rule
Flow Rule entries:
==================
@[153]: Channel [0]
IP src [0.0.0.0] mask [0x0]
IP dst [0.0.0.0] mask [0x0]
src port [0] mask [0x0]
dst port [5023] mask [0xffff] proto[TCP]
```

The above allows you to add a rule in the card driver which assures that every TCP packet destined to port 5001 (port used by my iperf tests) is sent to channel 0. The next step is to find which core in the CPU will be triggered by channel 0.

3) Identify which core inside a CPU handles the NIC interrupt:

Open a second terminal window and type:

/usr/bin/watch -d 'cat /proc/interrupts | grep eth4' (NOTICE the quotes here)

```
194: 3038081 63 0 423 2345 56 777 5555
 188 82 21118 PCI-MSI-X eth4_i0
```

Now, ping this interface eth4 from another system and come back to this screen, you will see the number "3038081" blinking and incrementing. This represents the number of packets that the interface eth4 is receiving, using the interrupt 194, and handled by core #0 (counting from the left after the "194" and starting from core #0 (3038081) then CPU#1 (63) and so on, until core #7 (5555). Now, we just found out that "194" is the interrupt used by the NIC handled by core#0, on CPU# 0.

To recap:

The above watch command allows us to see in real-time the incrementation of packets received on the interface on the core # that handles the NIC.

Now, let's discover how the cores are distributed in the CPU. To accomplish this, look for the phisycalID in the **/proc/cpuinfo** command output. At this point it depends on the architecture that

is available.

For this research, the two systems architecture tested and analyzed present two different approaches to handle memory for cores. Either the cores are sharing the L2 cache (Intel® Xeon® X5355) or each core has its own L2cache (AMD Opteron processors 8347). It is now crucial to identify which core is on CPU #0, 1, 2 and so forth.

As I have explained above:

In the DELL1950 each of the two dual quad-core Intel® Xeon® X5355 share the L2 cache of 4096 KBytes. In this architecture, core 0-2-4-6 are on CPU #0 (phisycalID 0) and 1-3-5-7 are on CPU #1 (physicalID1).

In the SUN Fire X4440 each core of the AMD 8347 Opteron processors has 512 KBytes of L2 cache not shared. In this architecture, cores #0-1-2-3 are on CPU #1, 4-5-6-7 on CPU #2, 8-9-10-11 on CPU #3, 12-13-14-15 on CPU#4.

Back to our example let's run:

```
less /proc/cpuinfo
[evalente@dell1950 proc]# more cpuinfo
processor : 0
vendor_id : GenuineIntel
cpu family: 6
model : 15
model name: Intel(R) Xeon(R) CPU X5355 @ 2.66GHz
stepping: 11
cpu MHz: 2660.031
cache size: 4096 KB
physical id: 1
siblings: 4
core id: 0
cpu cores: 4
fpu: yes
fpu_exception: yes
cpuid level: 10
wp: yes
```

```
flags: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr
pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht t
m syscall nx lm constant_tsc pni monitor ds_cpl vmx est tm2 cx16
xtpr lahf_lm
bogomips: 5323.73
clflush size: 64
cache_alignment: 64
address sizes: 38 bits physical, 48 bits virtual
power management:
processor: 1
vendor_id: GenuineIntel
cpu family: 6
model: 15
model name: Intel(R) Xeon(R) CPU X5355 @ 2.66GHz
stepping: 11
```

Look for the physicalID of the CPU that identifies which core is on which CPU. In our case, above the physicalID 1 indicates that core #0 (falsely named "processor" 0) is in the CPU #1 (phisicalID 1).

Now, identify all cores with the same physical id (this means they are on the same CPU). We are done with our investigation!

4) Use the "numactl" command to force the application (iperf) to run using a specific core as the following:

numactl --physcpubind=1 iperf -c ipaddressserver -p 5001 -m -t 30

The above command will run iperf bound to core #1. Again, the value to use for **--physcpubind** depends on the host architecture and a little experimenting.

This is what I have discovered: in the case of Intel CPUs, one of the cores identified at the end of step 3 - that does not handle the NIC interrupt and is located on the same CPU – has increased network throughput.

Two of the cores identified at the end of step 3, on the same

CPU, have slightly less throughput. The other four cores on the second CPU have even less performances.

In the case of AMD, the process is a little simpler: the core handling the NIC, and any other cores in the same CPU, will give you the best results. The other 12 cores on the other 3 CPUs will give the least results.

MYRICOM PCI Express- Myri-10GigE[6]

a) Check the driver version and make sure you have the latest from Myricom:

```
evalente@server]#ethtool -i eth4
driver: myri10ge
version: 1.0.0
firmware-version: 1.4.31 -P- 2008/04/25 17:26:57
bus-info: 0000:82:00.0
```

b) Customize memory settings buffers for TCP/IP and Core Tuning:

- Memory settings buffers suggested by Myricom for TCP/IP tuning:

```
net.ipv4.tcp_timestamps = 0
net.ipv4.tcp_sack = 0
net.ipv4.tcp_rmem = 4096 87380 16777216
net.ipv4.tcp_wmem = 4096 65536 16777216
```

- Memory settings buffers suggested by Myricom for Core tuning:

```
net.core.netdev_max_backlog = 250000
net.core.rmem_max = 16777216
net.core.wmem_max = 16777216
```

- Memory settings buffers for long distance (30 to 70 ms delay) and large data transfer (my tested and suggested settings):

TCP/IP memory tuning:

```
net.ipv4.tcp_timestamps = 0
net.ipv4.tcp_sack = 0
net.ipv4.tcp_rmem = 4096 134217728 134217728
```

```
net.ipv4.tcp_wmem = 4096 134217728 134217728
```

Core memory Tuning

```
net.core.netdev_max_backlog = 250000
net.core.rmem_max = 134217728
net.core.wmem_max = 134217728
```

To modify the above settings, add the commands to the **sysctl.conf** file, located in /etc folder, and then run:

```
/sbin/sysctl -p
```

By following this procedure, it will avoid the problem that, at next reboot, your system will reinitialize these settings to the default OS values. (I prefer this to the "echo" commands described earlier for the 1 GigE tuning).

c) Interrupt Coalescing

Myri10GE

"The Myri10GE driver is ethtool compliant, and you may adjust the interrupt coalescing parameter via

ethtool -C $DEVNAME rx-usecs $VALUE

The default setting is 25us for Linux kernels prior to 2.6.16, and is 75us for Linux kernels 2.6.16 and later. The default setting is a compromise between latency and CPU overhead."

I found that 100us is the best setting for this card for both architectures.

d) Check and eventually turn TCP segmentation OFF (same as the SUN card described above in subparagraph (d))

e) TXQUEUELEN at value 2500 (same as the SUN card described above in subparagraph (e))

f) CPU Binding/Affinity (or better: Core Binding/Affinity)

For this section, what was discovered with the SUN card settings is also valid for the Myricom card, except for step number 2 regarding the SUN channels.

The Myricom card has a different architecture and does not have channels like the SUN card.

It uses a low-latency kernel-bypass communication directly with applications that maximize the throughput of single stream connectivity.

Dell PowerEdge 1950 - best to bind the NIC card interrupt to one core and the Application interrupt to a different core in the same CPU.

To recap the steps for the Myricom here:

1) If it is running, IRQBALANCE service must be killed.

This allows us to get control of how interrupts are handled in the CPU (Core, L2cache sharing, internal bus).

2) Identify which core inside a CPU handles the NIC interrupt (like SUN card described above in subparagraph (f) bullet # 3).

4) Use the "numactl" command to force the application (iperf) to run using a specific core such as the following (like SUN card described above in subparagraph (f) bullet # 4).

IMPORTANT for both cards*

The physical ID and the CPU interrupt binding (NIC or other processes) CHANGE every time the system reboots or you remove the module (driver).

My recommendation is to proceed by checking every time you need to perform a network test or a large data transfer!

End-to-End 10 Gigabit Network Tests

So far, I have tested several types of cards (including PCI X) and several system architectures. Below I present the network performance results of the two fastest types of NIC cards and the

two fastest types of systems architectures.

All my WAN tests have San Diego, CA as the source endpoint. Iperf has been used to send a single stream at 10 seconds (default) as well as 30, 60, and 90-second tests. In addition, a test of 5 parallel streams using the option "-P" with the interval of 30-60-90 seconds has been performed.

In the LAN testing, the peak (best throughput) of a 3-sample test has been recorded and a unidirectional test has been performed from each system.

In the WAN testing, the option "-r" (tradeoff) in iperf has been used, which generates two separate bidirectional tests individually (not simultaneously). The peak of the 3-sample test has been picked from each direction and the average of the two values has been recorded for each type of test.

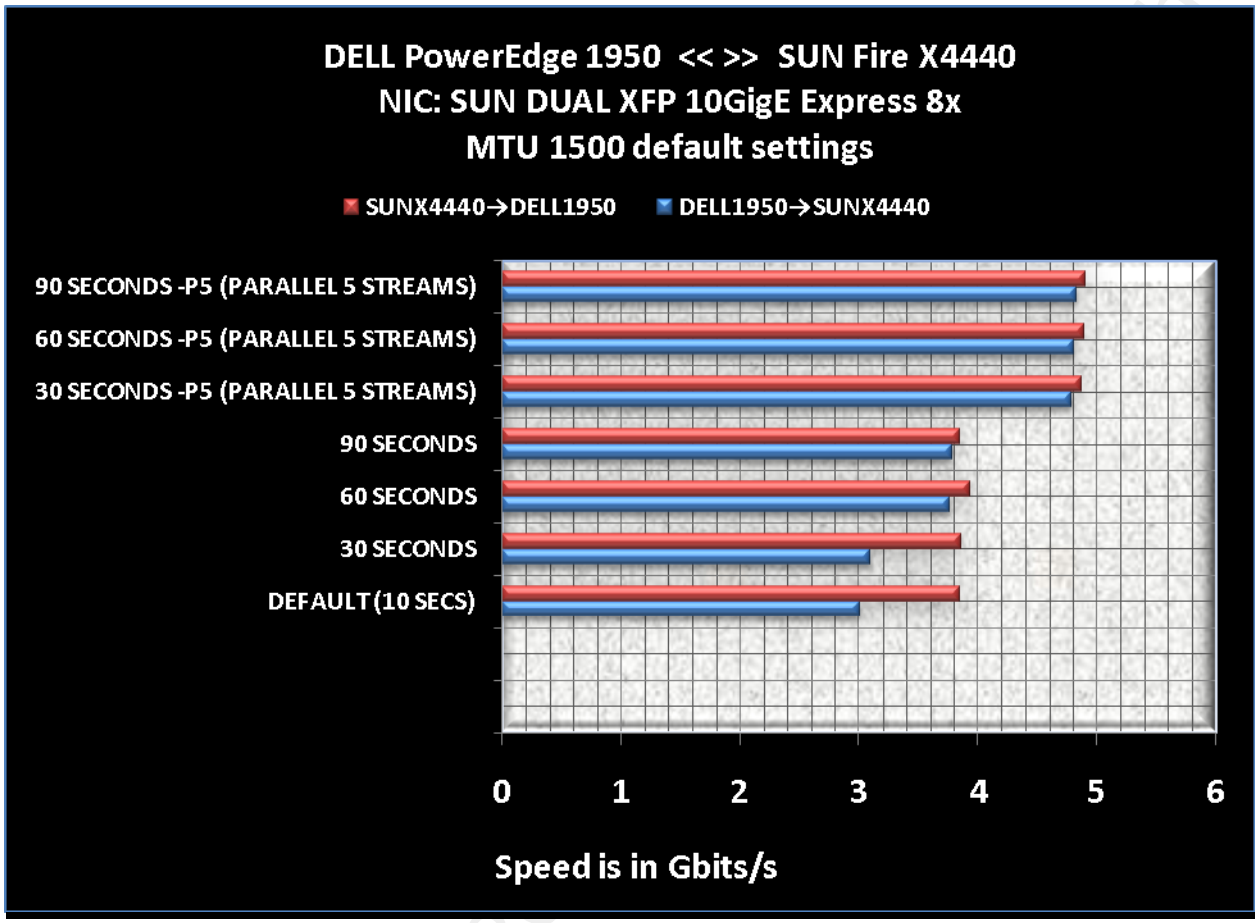
Testing starts with the default settings and mtu equal to 1500, then after tuning the settings and change the mtu to 9000, I recorded the best performances (highest throughput) for each type of test.

LAN TESTING RESULTS

DELL PowerEdge 1950=DELL1950 << >> SUN Fire X4440=SUNX4440 - Speed is in Gbits/s - P=Parallel streams

NIC: SUN DUAL XFP 10GigE Express 8x				
IPERF2.0.2	DELL1950→SUNX4440		SUNX4440→DELL1950	
	SINGLE STREAM	P5	SINGLE STREAM	P5
MTU 1500 default settings				
DEFAULT (10 SECS)	3.01		3.85	
30 SECONDS	3.10		3.86	
60 SECONDS	3.76		3.93	
90 SECONDS	3.78		3.85	
30 SECONDS -P5 (PARALLEL 5 STREAMS)		4.79		4.87
60 SECONDS -P5 (PARALLEL 5 STREAMS)		4.81		4.89
90 SECONDS -P5 (PARALLEL 5 STREAMS)		4.83		4.90

Table 11

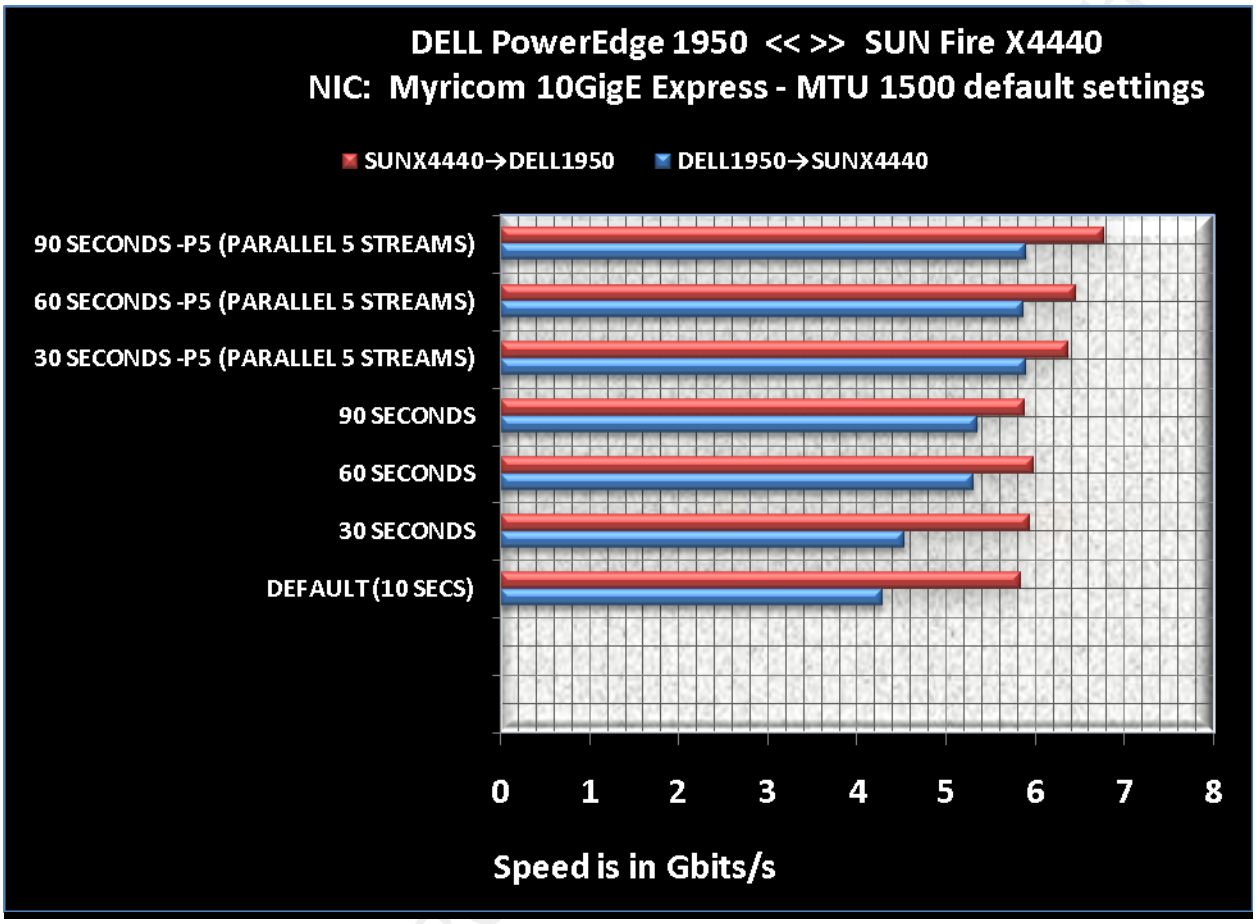


Graph A

DELL PowerEdge 1950=DELL1950 << >> SUN Fire X4440=SUNX4440 - Speed is in Gbits/s - P=Parallel streams

NIC: Myricom 10GigE Express				
IPERF2.0.2	DELL1950→SUNX4440		SUNX4440→DELL1950	
	SINGLE STREAM	P5	SINGLE STREAM	P5
MTU 1500 default settings				
DEFAULT (10 SECS)	4.28		5.84	
30 SECONDS	4.53		5.94	
60 SECONDS	5.30		5.98	
90 SECONDS	5.35		5.88	
30 SECONDS -P5 (PARALLEL 5 STREAMS)		5.89		6.36
60 SECONDS -P5 (PARALLEL 5 STREAMS)		5.87		6.45
90 SECONDS -P5 (PARALLEL 5 STREAMS)		5.90		6.77

Table 12

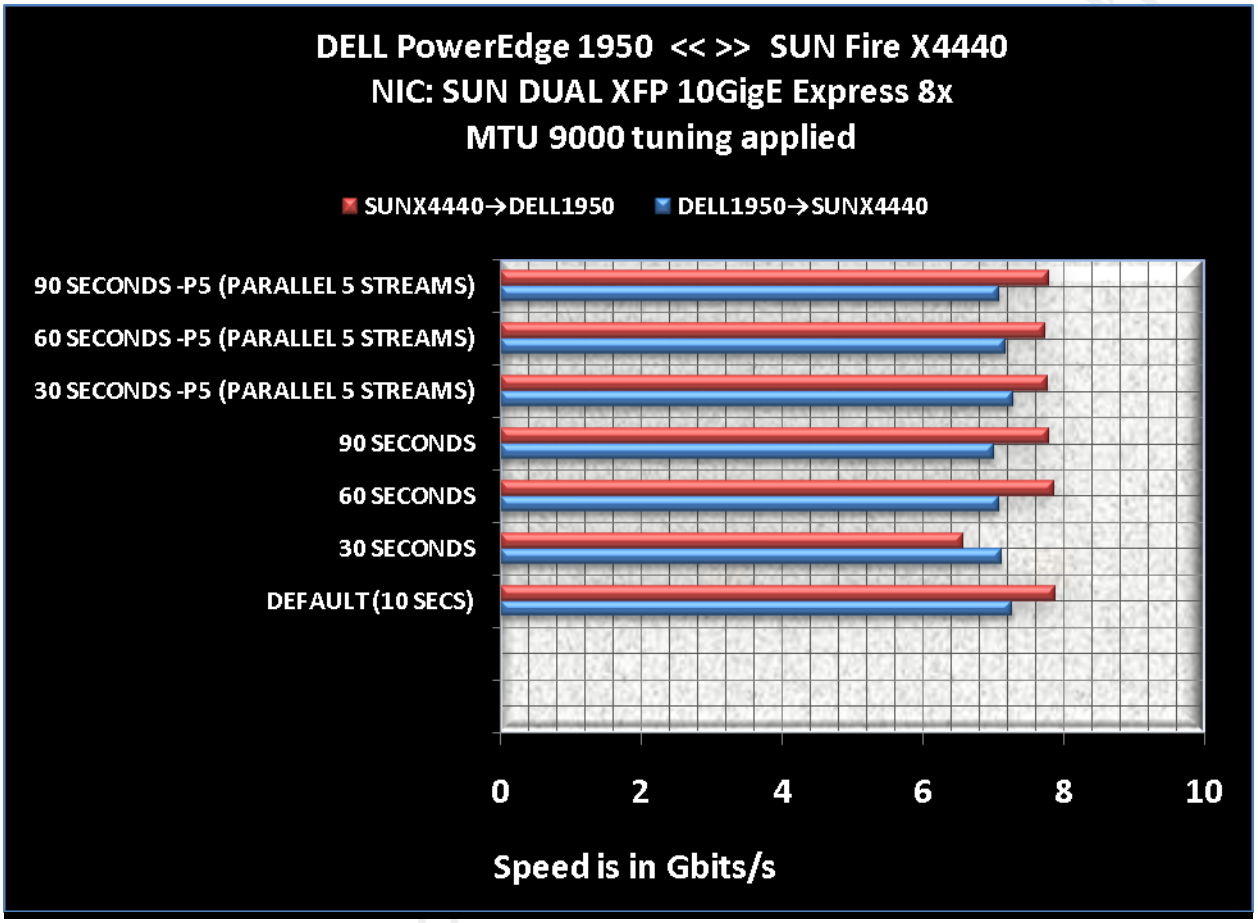


Graph B

DELL PowerEdge 1950=DELL1950 << >> SUN Fire X4440=SUNX4440 - Speed is in Gbits/s - P=Parallel streams

NIC: SUN DUAL XFP 10GigE Express 8x				
IPERF2.0.2	DELL1950→SUNX4440		SUNX4440→DELL1950	
	SINGLE STREAM	P5	SINGLE STREAM	P5
MTU 9000 tuning applied				
DEFAULT (10 SECS)	7.26		7.89	
30 SECONDS	7.11		6.57	
60 SECONDS	7.08		7.86	
90 SECONDS	7.01		7.80	
30 SECONDS -P5 (PARALLEL 5 STREAMS)		7.29		7.78
60 SECONDS -P5 (PARALLEL 5 STREAMS)		7.17		7.73
90 SECONDS -P5 (PARALLEL 5 STREAMS)		7.08		7.80

Table 13

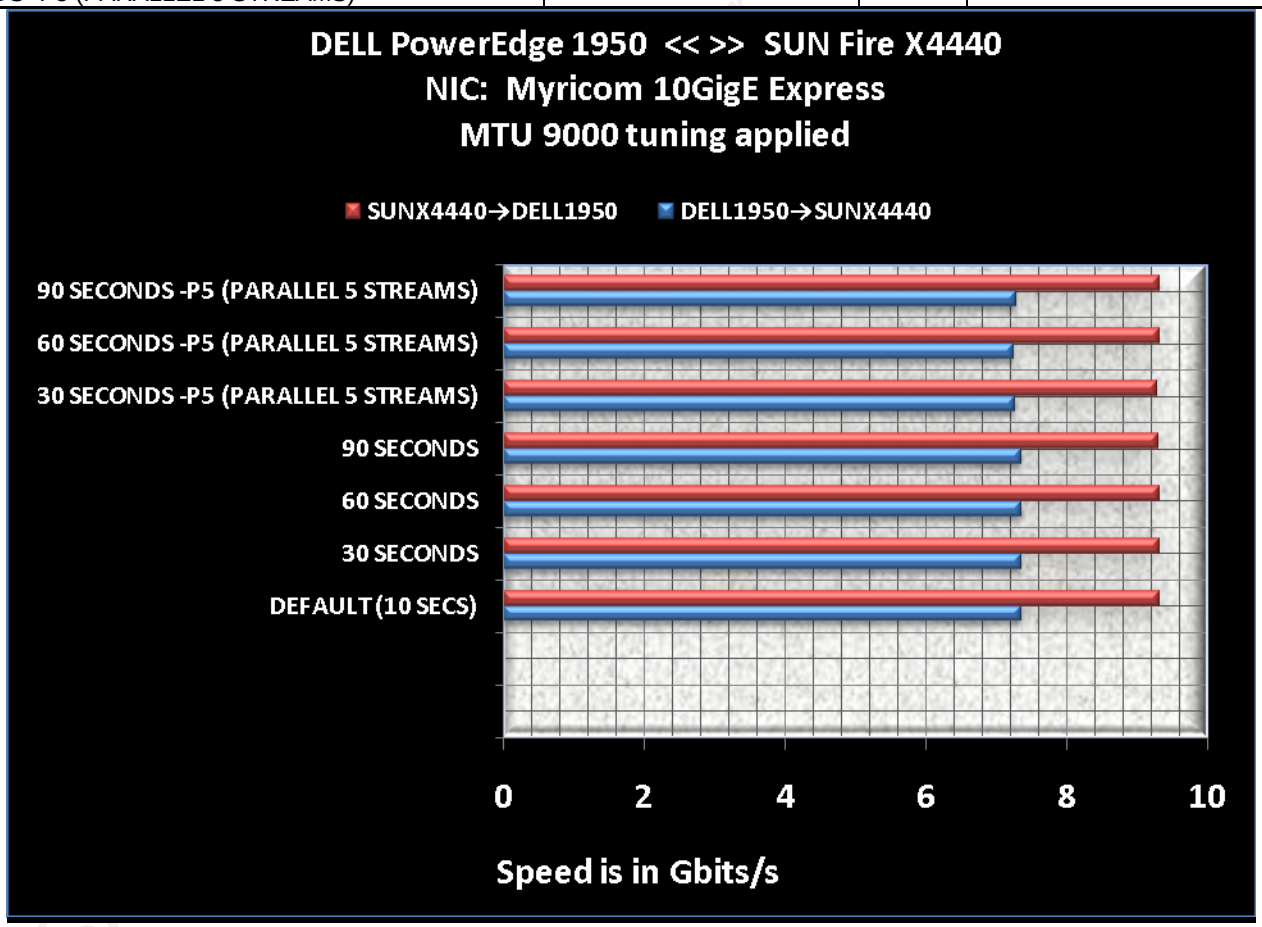


Graph C

Table 14

DELL PowerEdge 1950=DELL1950 << >> SUN Fire X4440=SUNX4440 - Speed is in Gbits/s - P=Parallel streams

Myricom 10GigE Express				
IPERF2.0.2	DELL1950→SUNX4440		SUNX4440→DELL1950	
	SINGLE STREAM	P5	SINGLE STREAM	P5
MTU 9000 tuning applied				
DEFAULT (10 SECS)	7.35		9.32	
30 SECONDS	7.35		9.32	
60 SECONDS	7.35		9.32	
90 SECONDS	7.35		9.31	
30 SECONDS -P5 (PARALLEL 5 STREAMS)		7.26		9.29
60 SECONDS -P5 (PARALLEL 5 STREAMS)		7.25		9.32
90 SECONDS -P5 (PARALLEL 5 STREAMS)		7.29		9.32

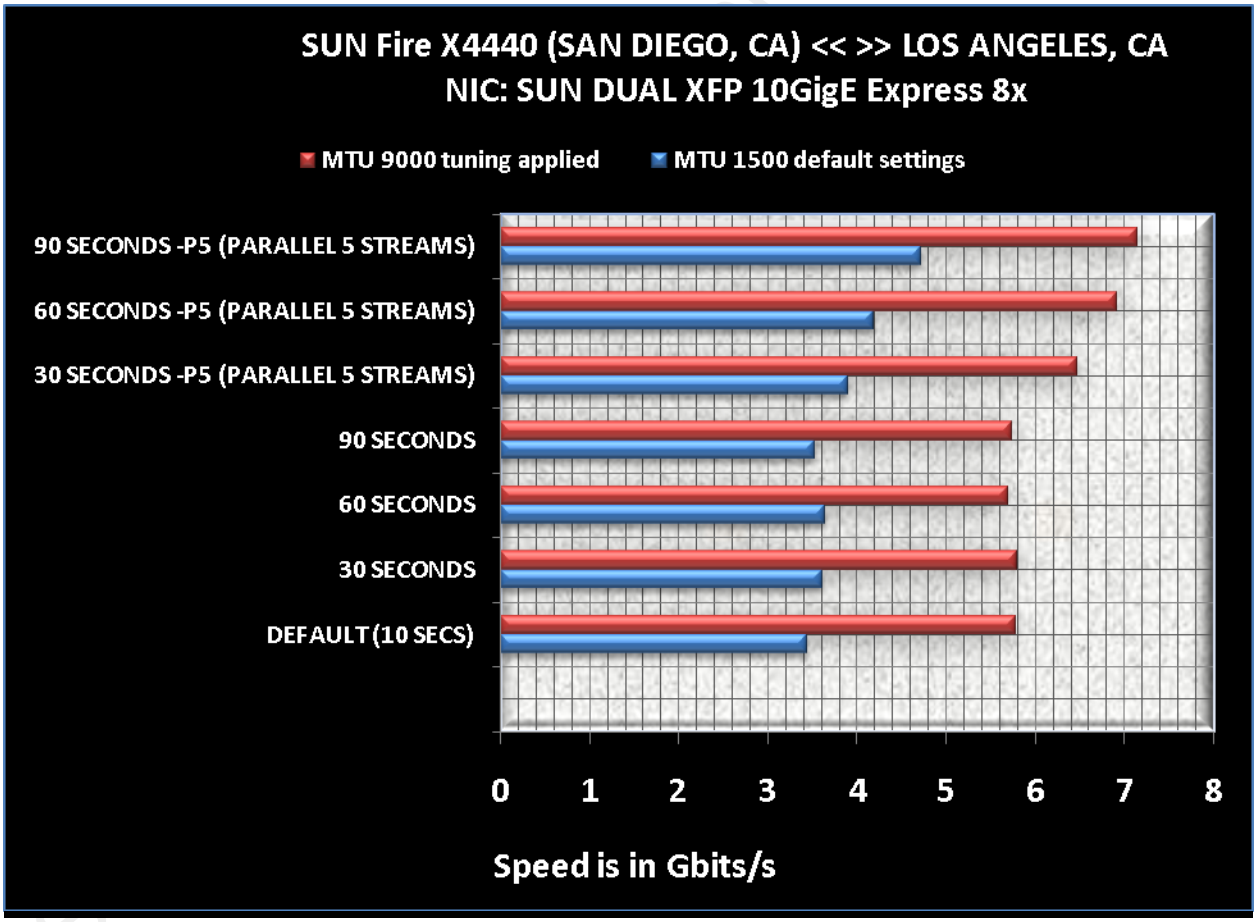


Graph D

SUN Fire X4440 (SAN DIEGO, CA) << >> LOS ANGELES, CA - Speed is in Gbits/s - P=Parallel streams

NIC: SUN DUAL XFP 10GigE Express 8x				
IPERF2.0.2	MTU 1500 default settings		MTU 9000 tuning applied	
	SINGLE STREAM (average)	P5 (average)	SINGLE STREAM (average)	P5 (average)
DEFAULT (10 SECS)	3.44		5.78	
30 SECONDS	3.61		5.79	
60 SECONDS	3.64		5.69	
90 SECONDS	3.53		5.73	
30 SECONDS -P5 (PARALLEL 5 STREAMS)		3.90		6.47
60 SECONDS -P5 (PARALLEL 5 STREAMS)		4.19		6.91
90 SECONDS -P5 (PARALLEL 5 STREAMS)		4.72		7.14

Table 15

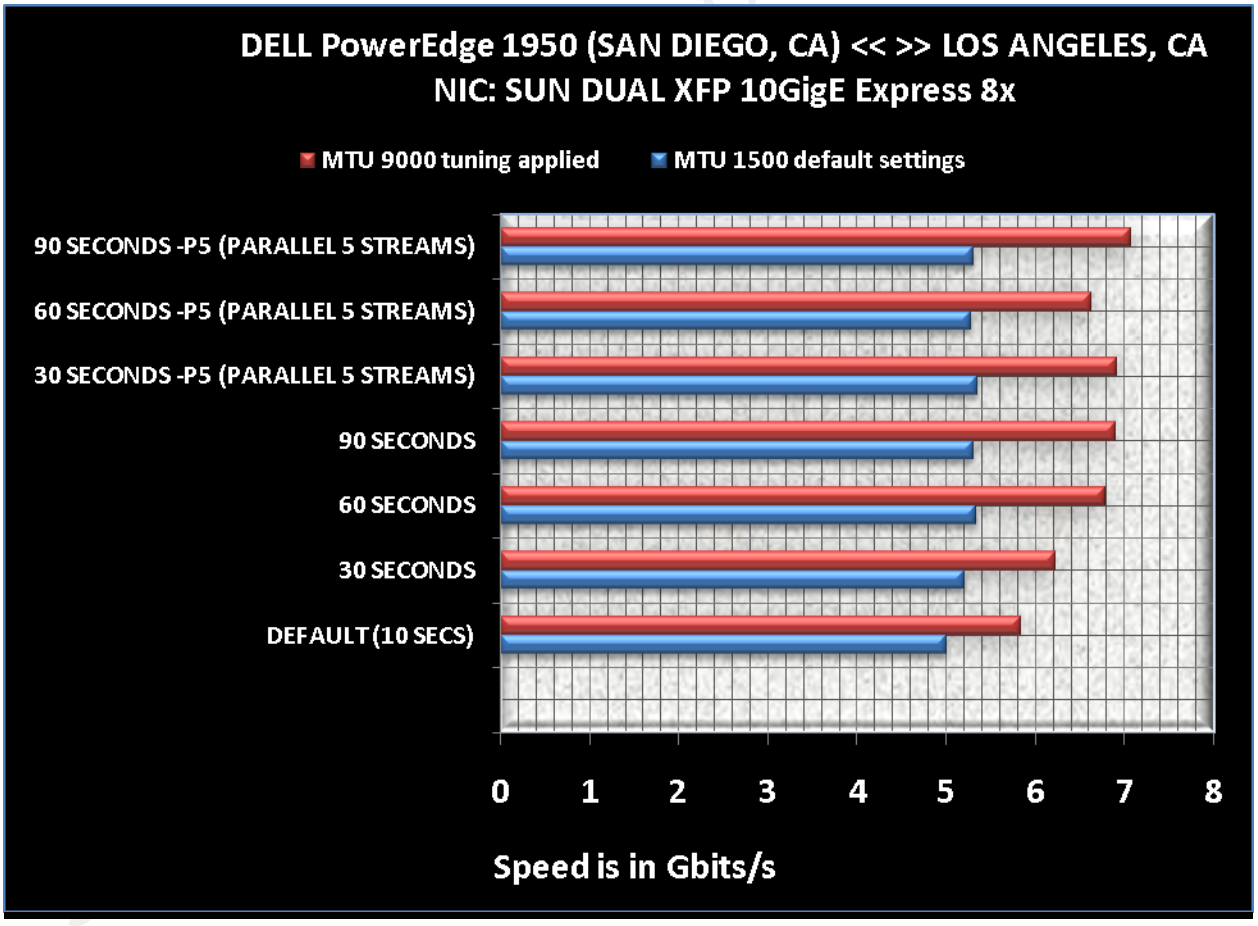


Graph E

DELL PowerEdge 1950 (SAN DIEGO, CA) << >> LOS ANGELES, CA - Speed is in Gbits/s - P=Parallel streams

NIC: SUN DUAL XFP 10GigE Express 8x				
IPERF2.0.2	MTU 1500 default settings		MTU 9000 tuning applied	
	SINGLE STREAM (average)	P5 (average)	SINGLE STREAM (average)	P5 (average)
DEFAULT (10 SECS)	5.00		5.84	
30 SECONDS	5.21		6.23	
60 SECONDS	5.33		6.79	
90 SECONDS	5.30		6.90	
30 SECONDS -P5 (PARALLEL 5 STREAMS)		5.35		6.92
60 SECONDS -P5 (PARALLEL 5 STREAMS)		5.27		6.63
90 SECONDS -P5 (PARALLEL 5 STREAMS)		5.31		7.07

Table 16

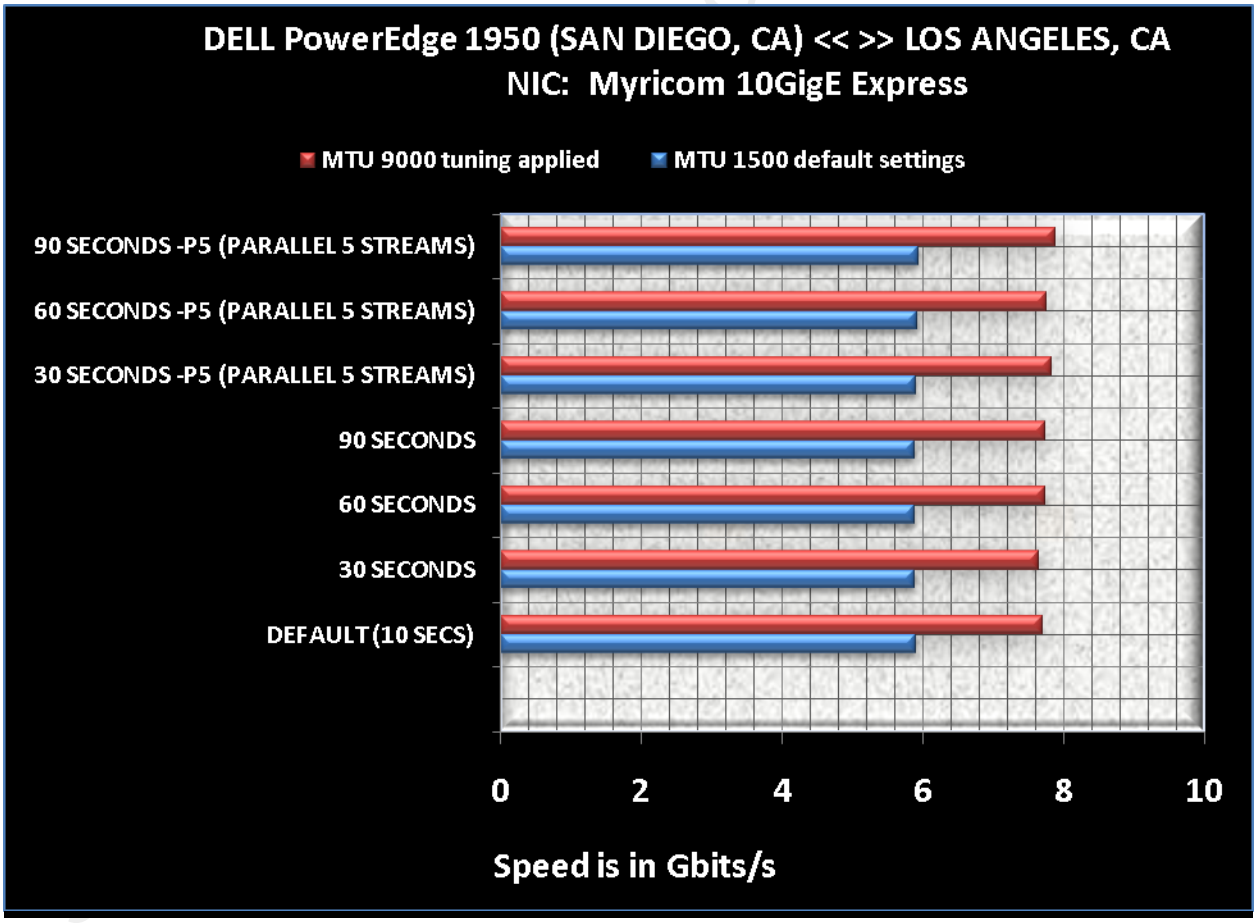


Graph F

DELL PowerEdge 1950 (SAN DIEGO, CA) << >> LOS ANGELES, CA - Speed is in Gbits/s - P=Parallel streams

Myricom 10GigE Express				
IPERF2.0.2	MTU 1500 default settings		MTU 9000 tuning applied	
	SINGLE STREAM (average)	P5 (average)	SINGLE STREAM (average)	P5 (average)
DEFAULT (10 SECS)	5.90		7.71	
30 SECONDS	5.89		7.65	
60 SECONDS	5.88		7.74	
90 SECONDS	5.89		7.74	
30 SECONDS -P5 (PARALLEL 5 STREAMS)		5.90		7.83
60 SECONDS -P5 (PARALLEL 5 STREAMS)		5.92		7.75
90 SECONDS -P5 (PARALLEL 5 STREAMS)		5.94		7.88

Table 17

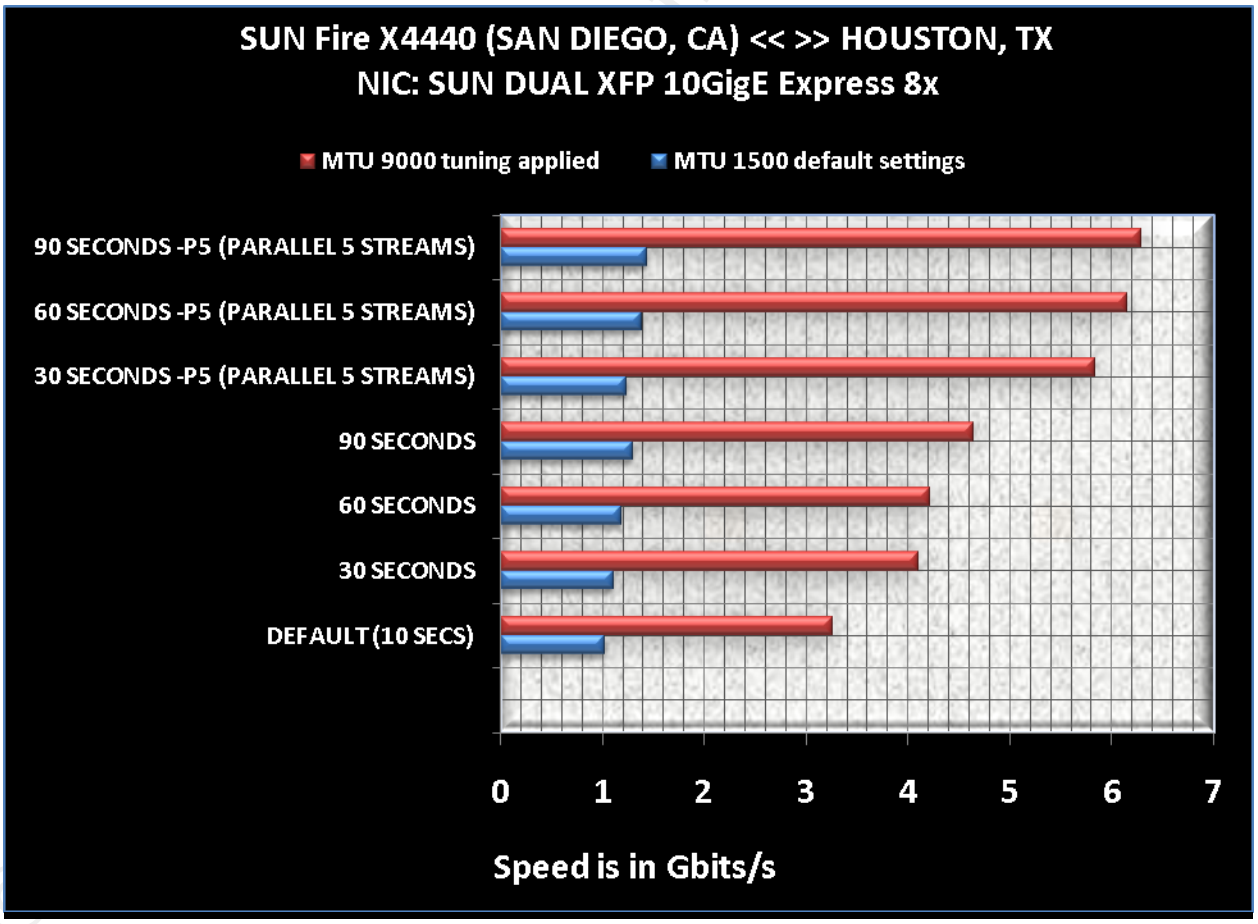


Graph G

SUN Fire X4440 (SAN DIEGO, CA) << >> HOUSTON, TX - Speed is in Gbits/s - P=Parallel streams

NIC: SUN DUAL XFP 10GigE Express 8x				
IPERF2.0.2	MTU 1500 default settings		MTU 9000 tuning applied	
	SINGLE STREAM (average)	P5 (average)	SINGLE STREAM (average)	P5 (average)
DEFAULT (10 SECS)	1.02		3.26	
30 SECONDS	1.11		4.10	
60 SECONDS	1.18		4.21	
90 SECONDS	1.30		4.64	
30 SECONDS -P5 (PARALLEL 5 STREAMS)		1.23		5.84
60 SECONDS -P5 (PARALLEL 5 STREAMS)		1.28		6.15
90 SECONDS -P5 (PARALLEL 5 STREAMS)		1.44		6.29

Table 18

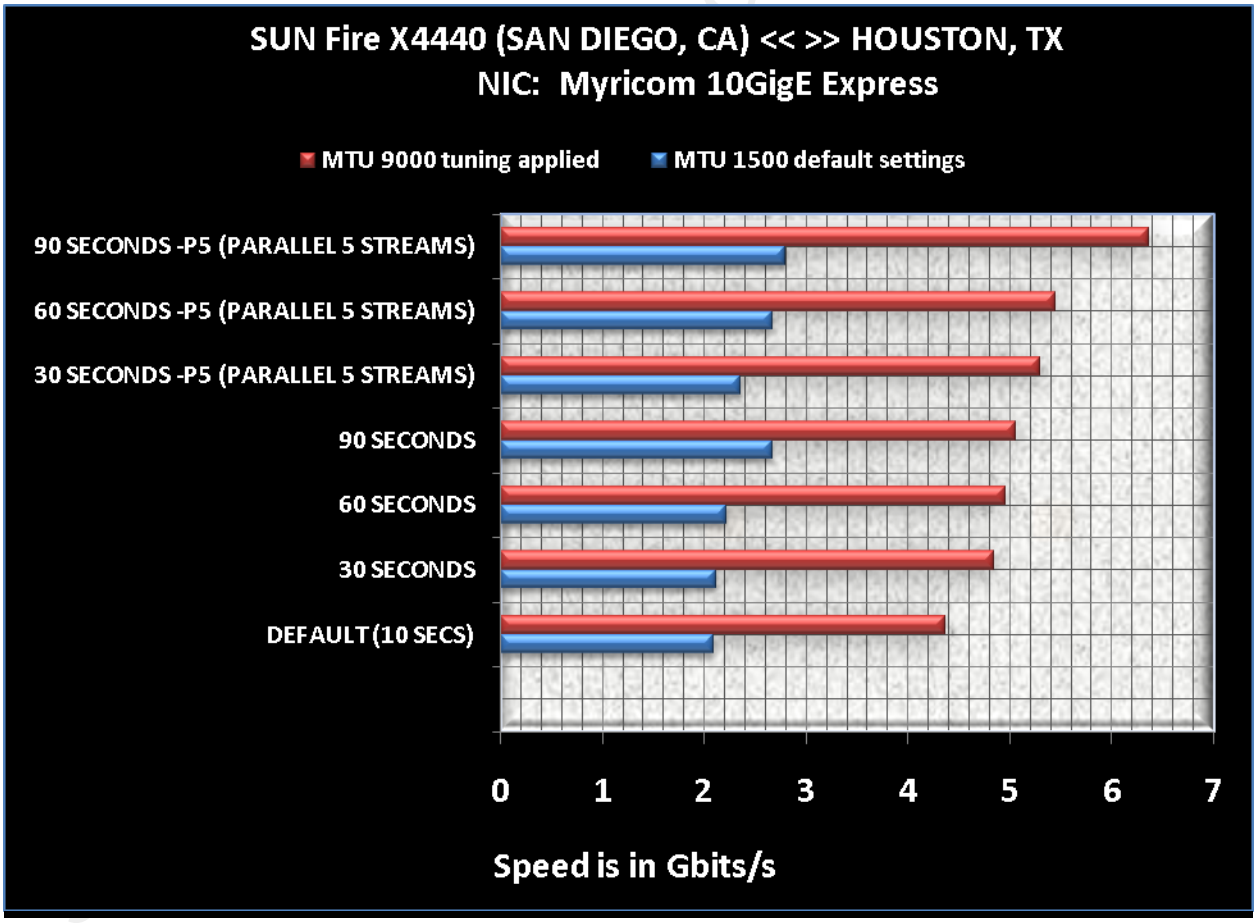


Graph H

SUN Fire X4440 (SAN DIEGO, CA) << >> HOUSTON, TX - Speed is in Gbits/s - P=Parallel streams

NIC: Myricom 10GigE Express				
IPERF2.0.2	MTU 1500 default settings		MTU 9000 tuning applied	
	SINGLE STREAM (average)	P5 (average)	SINGLE STREAM (average)	P5 (average)
DEFAULT (10 SECS)	2.09		4.36	
30 SECONDS	2.12		4.84	
60 SECONDS	2.21		4.95	
90 SECONDS	2.67		5.05	
30 SECONDS -P5 (PARALLEL 5 STREAMS)		2.35		5.30
60 SECONDS -P5 (PARALLEL 5 STREAMS)		2.67		5.45
90 SECONDS -P5 (PARALLEL 5 STREAMS)		2.80		6.36

Table 19

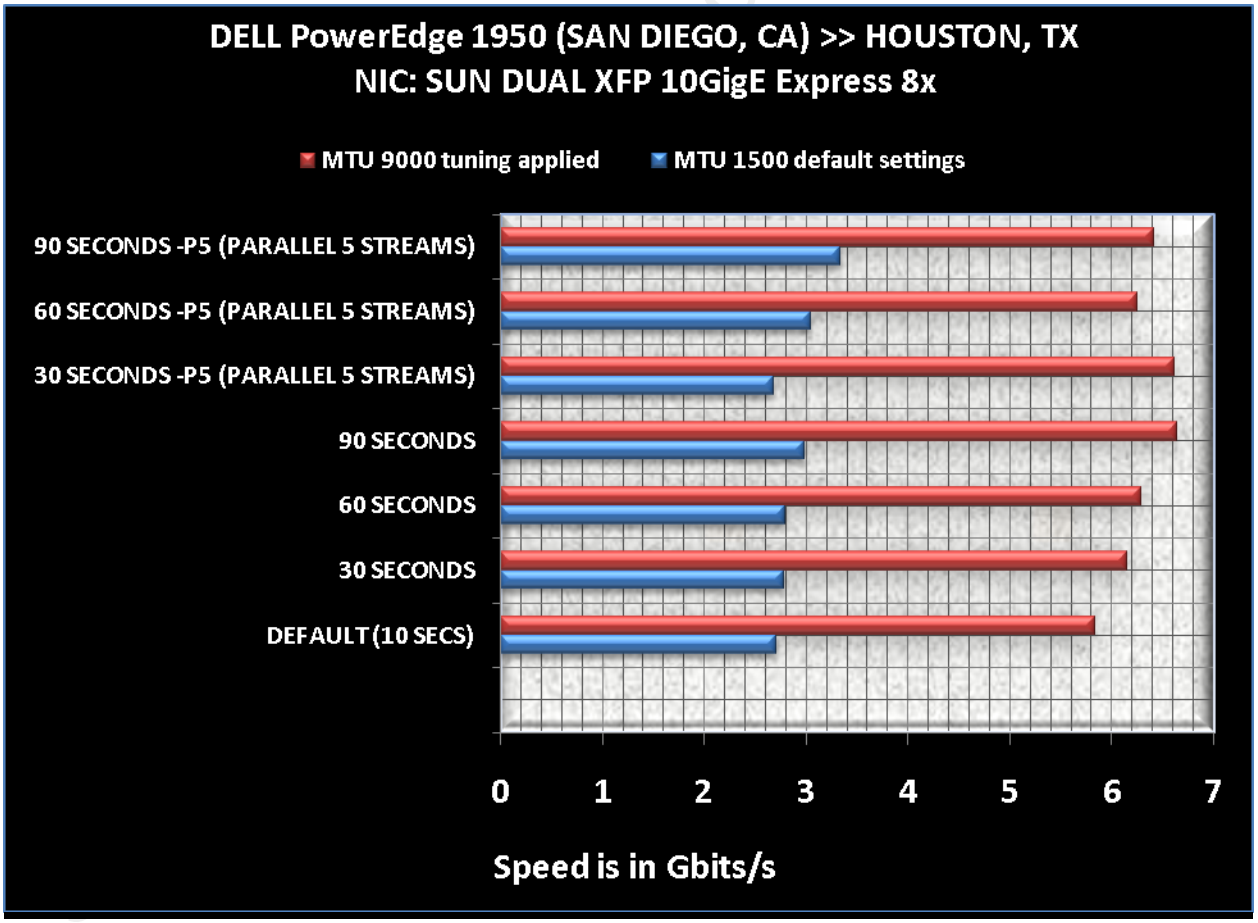


Graph I

DELL PowerEdge 1950 (SAN DIEGO, CA) << >> HOUSTON, TX - Speed is in Gbits/s - P=Parallel streams

NIC: SUN DUAL XFP 10GigE Express 8x				
IPERF2.0.2	MTU 1500 default settings		MTU 9000 tuning applied	
	SINGLE STREAM (average)	P5 (average)	SINGLE STREAM (average)	P5 (average)
DEFAULT (10 SECS)	2.70		5.84	
30 SECONDS	2.78		6.15	
60 SECONDS	2.80		6.29	
90 SECONDS	2.98		6.64	
30 SECONDS -P5 (PARALLEL 5 STREAMS)		2.68		6.61
60 SECONDS -P5 (PARALLEL 5 STREAMS)		3.05		6.25
90 SECONDS -P5 (PARALLEL 5 STREAMS)		3.33		6.41

Table 20

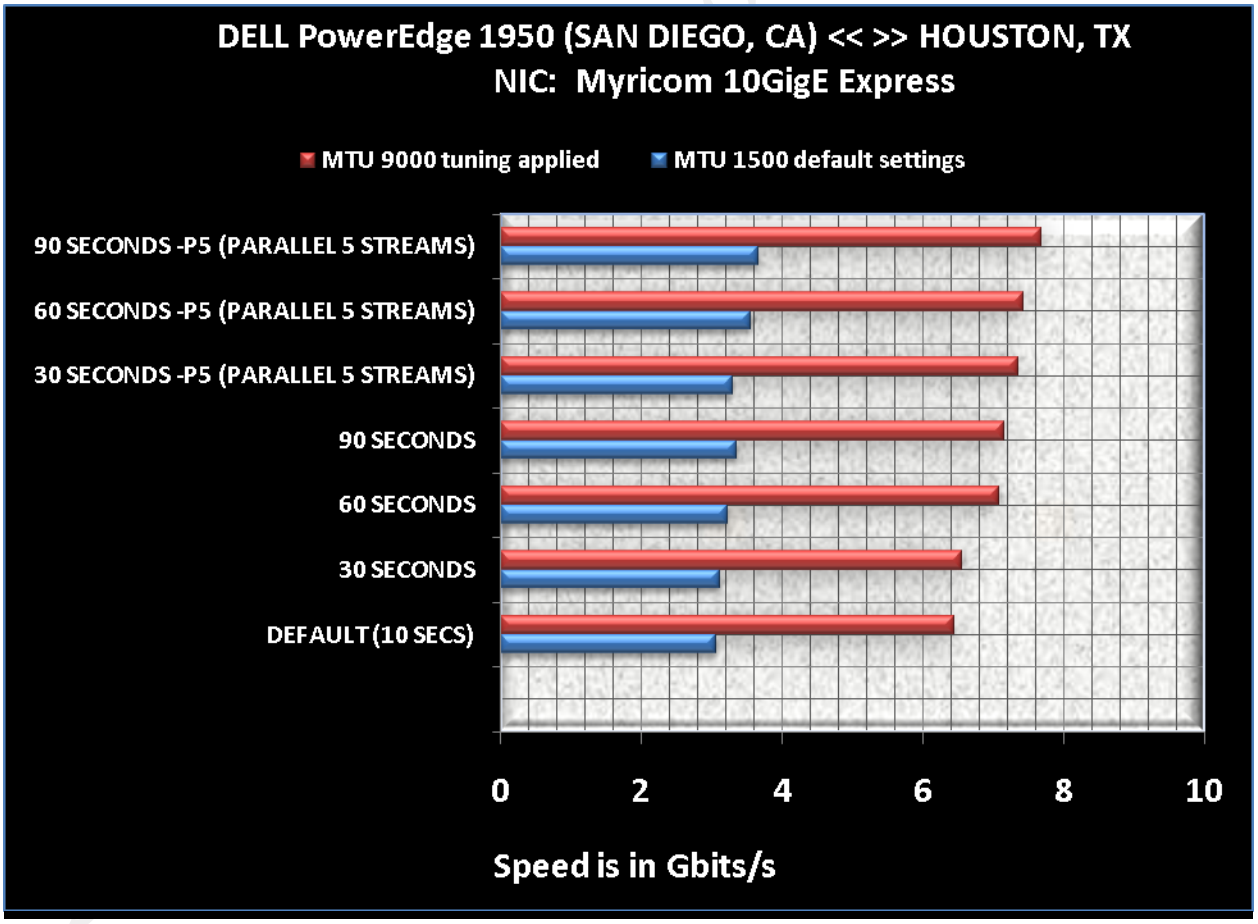


Graph J

DELL PowerEdge 1950 (SAN DIEGO, CA) << >> HOUSTON, TX - Speed is in Gbits/s - P=Parallel streams

Myricom 10GigE Express				
IPERF2.0.2	MTU 1500 default settings		MTU 9000 tuning applied	
	SINGLE STREAM (average)	P5 (average)	SINGLE STREAM (average)	P5 (average)
DEFAULT (10 SECS)	3.06		6.45	
30 SECONDS	3.12		6.56	
60 SECONDS	3.22		7.09	
90 SECONDS	3.35		7.15	
30 SECONDS -P5 (PARALLEL 5 STREAMS)		3.30		7.35
60 SECONDS -P5 (PARALLEL 5 STREAMS)		3.56		7.43
90 SECONDS -P5 (PARALLEL 5 STREAMS)		3.67		7.68

Table 21

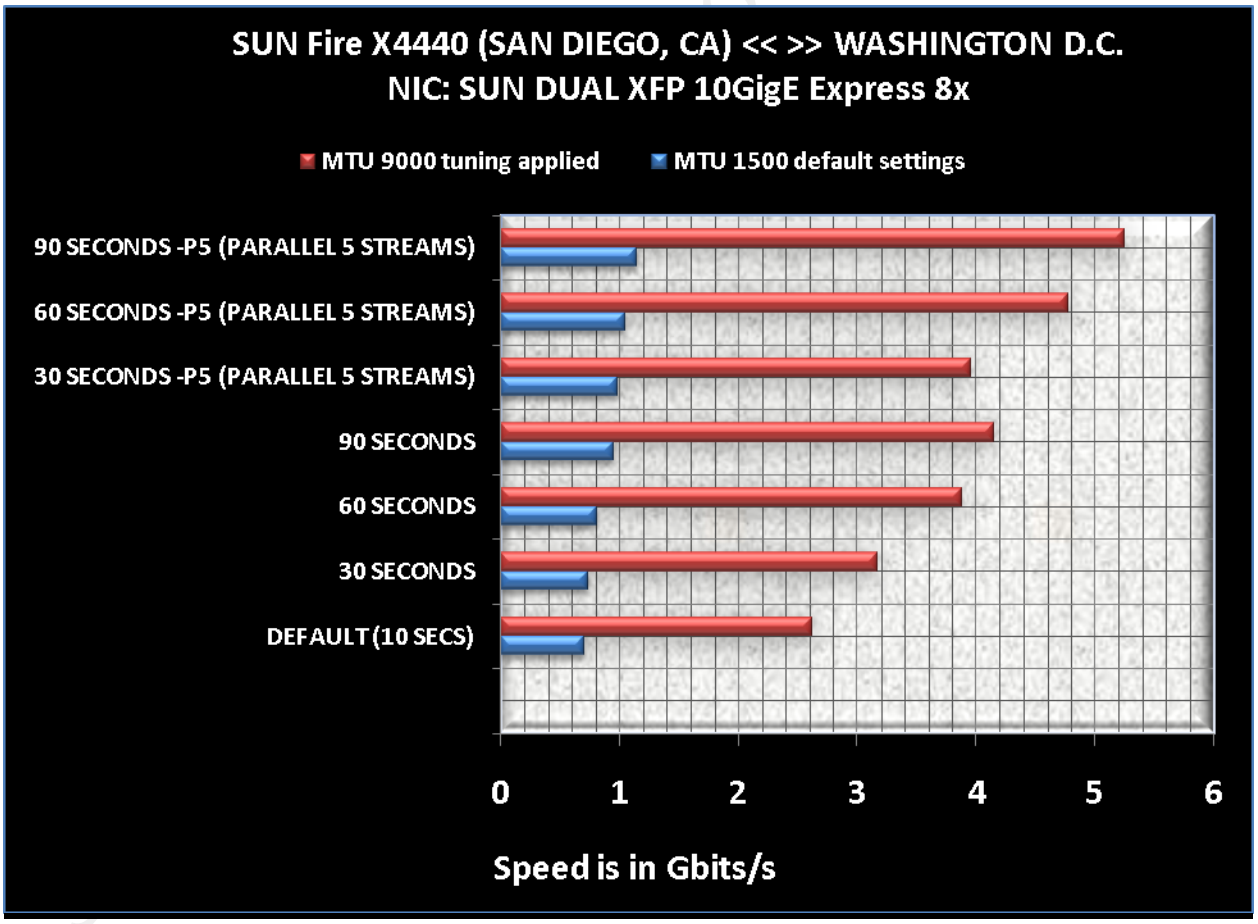


Graph K

SUN Fire X4440 (SAN DIEGO, CA) << >> WASHINGTON D.C. - Speed is in Gbits/s - P=Parallel streams

NIC: SUN DUAL XFP 10GigE Express 8x				
IPERF2.0.2	MTU 1500 default settings		MTU 9000 tuning applied	
	SINGLE STREAM (average)	P5 (average)	SINGLE STREAM (average)	P5 (average)
DEFAULT (10 SECS)	0.71		2.62	
30 SECONDS	0.74		3.17	
60 SECONDS	0.81		3.88	
90 SECONDS	0.96		4.15	
30 SECONDS -P5 (PARALLEL 5 STREAMS)		0.99		3.96
60 SECONDS -P5 (PARALLEL 5 STREAMS)		1.05		4.78
90 SECONDS -P5 (PARALLEL 5 STREAMS)		1.15		5.25

Table 22

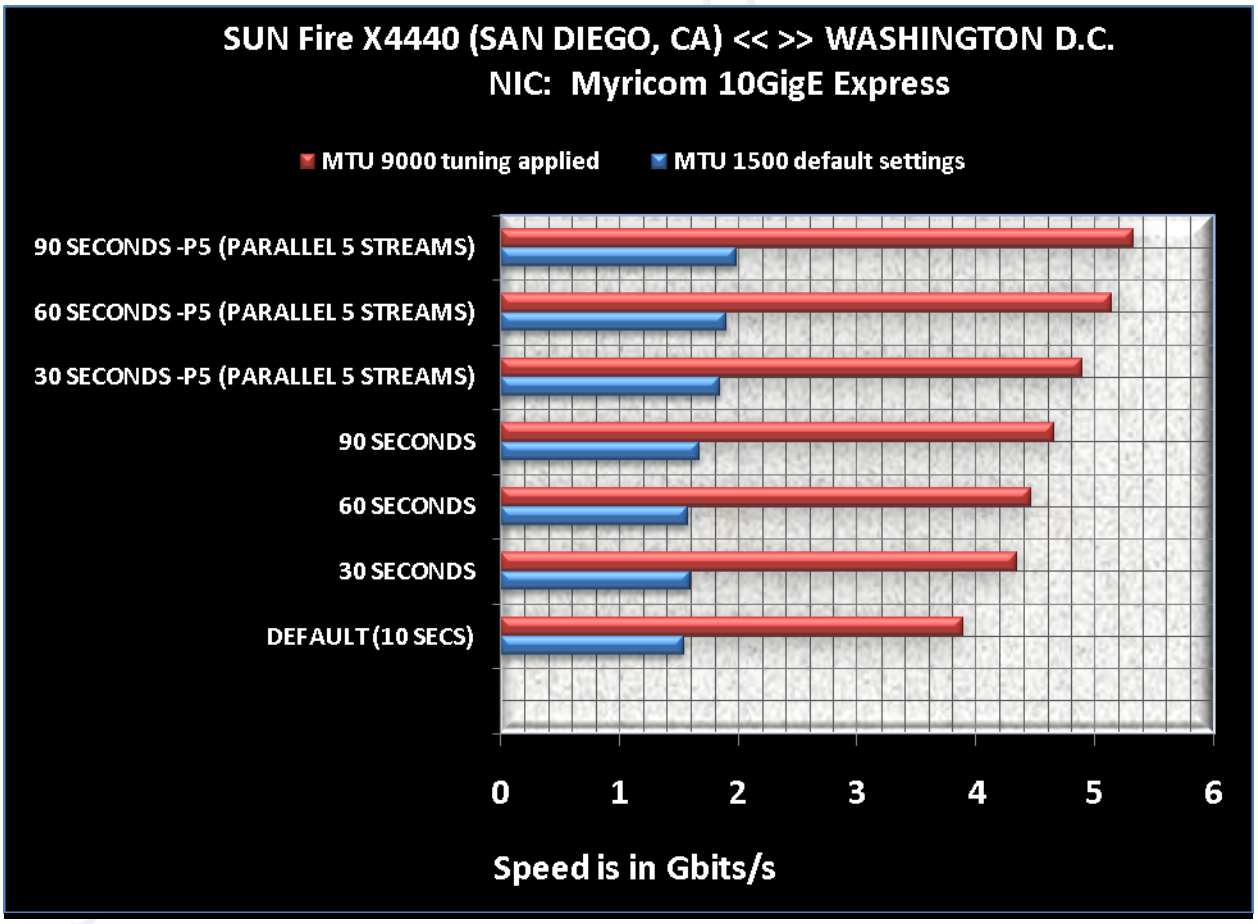


Graph L

SUN Fire X4440 (SAN DIEGO, CA) << >> WASHINGTON D.C. - Speed is in Gbits/s - P=Parallel streams

NIC: Myricom 10GigE Express				
IPERF2.0.2	MTU 1500 default settings		MTU 9000 tuning applied	
	SINGLE STREAM (average)	P5 (average)	SINGLE STREAM (average)	P5 (average)
DEFAULT (10 SECS)	1.54		3.89	
30 SECONDS	1.60		4.34	
60 SECONDS	1.58		4.46	
90 SECONDS	1.67		4.66	
30 SECONDS -P5 (PARALLEL 5 STREAMS)		1.84		4.89
60 SECONDS -P5 (PARALLEL 5 STREAMS)		1.90		5.14
90 SECONDS -P5 (PARALLEL 5 STREAMS)		1.98		5.32

Table 23

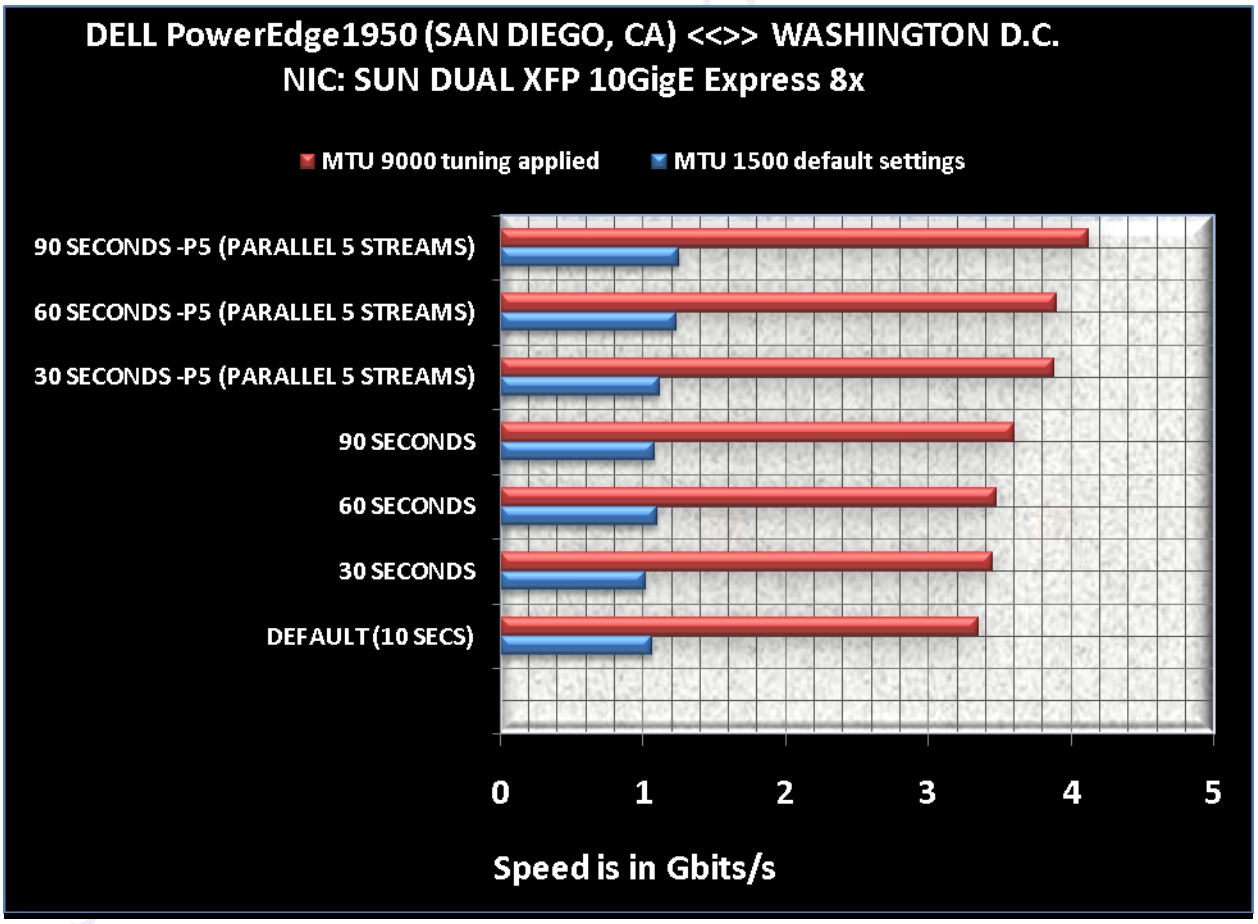


Graph M

DELL PowerEdge 1950 (SAN DIEGO, CA) << >> WASHINGTON D.C. - Speed is in Gbits/s - P=Parallel streams

NIC: SUN DUAL XFP 10GigE Express 8x				
IPERF2.0.2	MTU 1500 default settings		MTU 9000 tuning applied	
	SINGLE STREAM (average)	P5 (average)	SINGLE STREAM (average)	P5 (average)
DEFAULT (10 SECS)	1.06		3.35	
30 SECONDS	1.02		3.45	
60 SECONDS	1.10		3.48	
90 SECONDS	1.08		3.60	
30 SECONDS -P5 (PARALLEL 5 STREAMS)		1.12		3.88
60 SECONDS -P5 (PARALLEL 5 STREAMS)		1.23		3.90
90 SECONDS -P5 (PARALLEL 5 STREAMS)		1.25		4.12

Table 24

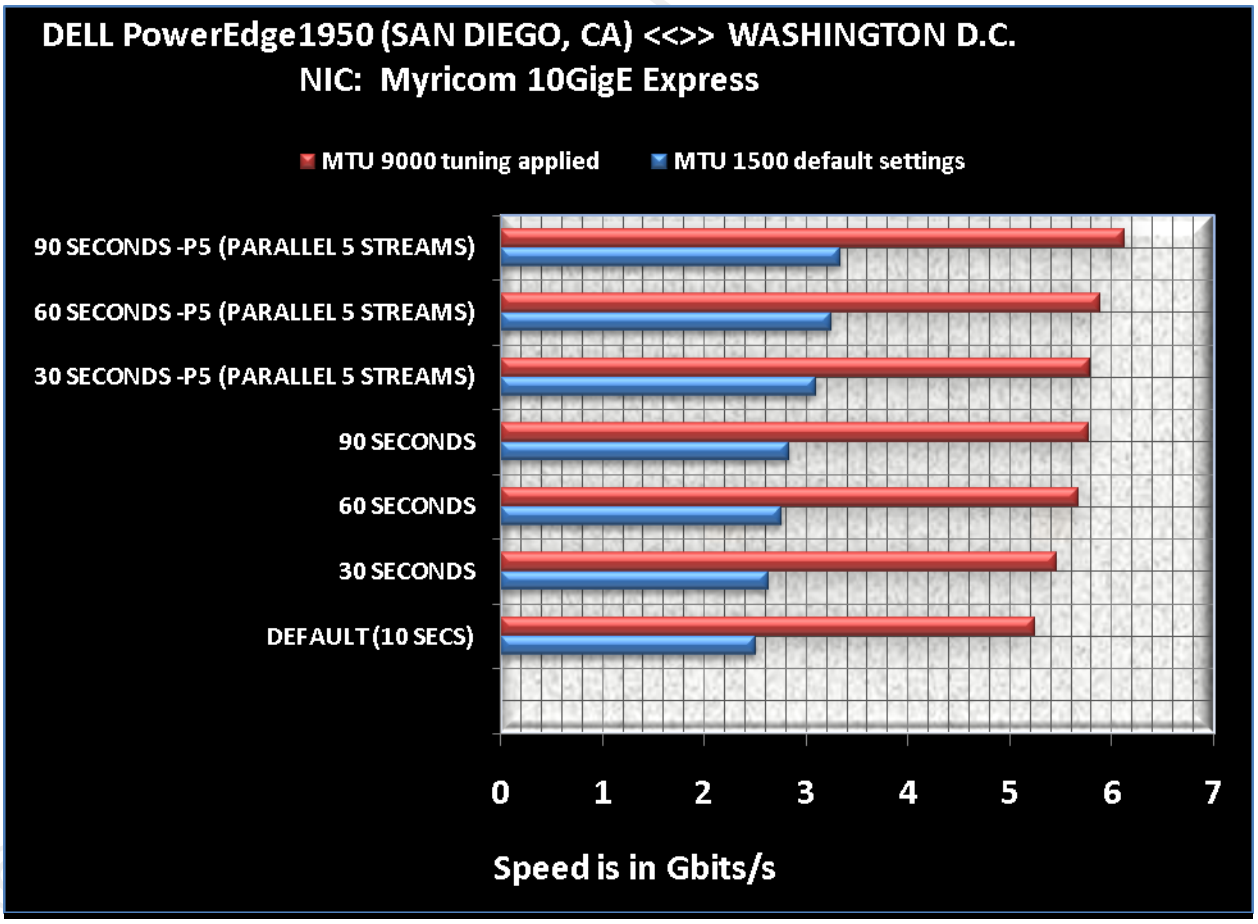


Graph N

DELL PowerEdge 1950 (SAN DIEGO, CA) << >> WASHINGTON D.C. - Speed is in Gbits/s - P=Parallel streams

Myricom 10GigE Express				
IPERF2.0.2	MTU 1500 default settings		MTU 9000 tuning applied	
	SINGLE STREAM (average)	P5 (average)	SINGLE STREAM (average)	P5 (average)
DEFAULT (10 SECS)	2.51		5.25	
30 SECONDS	2.63		5.46	
60 SECONDS	2.76		5.67	
90 SECONDS	2.83		5.77	
30 SECONDS -P5 (PARALLEL 5 STREAMS)		3.10		5.78
60 SECONDS -P5 (PARALLEL 5 STREAMS)		3.25		5.88
90 SECONDS -P5 (PARALLEL 5 STREAMS)		3.34		6.12

Table 25



Graph O

Network Performance Findings

Network TCP/IP stack tuning in LAN and back-to-back tests have no relevant effect. Conversely, in long-distance tests the following has been observed.

1GigE Network Performance Testing:

The delay between San Diego and the two sites under testing is: 62.7 ms for Chicago and 80.3 ms for New York (see below table 6 and 7). After tuning, there was an increase of roughly 40% for Chicago and 33% for New York in network throughput. The original test with default settings showed values of 584 and 535 Mbits/s to Chicago and New York respectively. The second tests, after tuning, showed 822 and 793 Megabits respectively.

Traceroute from San Diego to Chicago (xxx.xx.xxx.xxx), 30 hops max, 46 byte packets

```
1 xx.xxx.xxx.xx (xx.xxx.xxx.xx) 0.325 ms 0.209 ms 0.237 ms
2 xxx-xxx-xxx xx.xxx (xxx.xx.xxx.xxx) 6.575 ms 6.408 ms 6.431 ms
3 xxx-xxx-x-x xx.xxx (xxx.xxx.xxx.xxx) 6.319 ms 6.319 ms 6.327 ms
4 xxx-xxx-x-x xx.xxx (xxx.xx.xxx.xx) 38.525 ms 38.311 ms 38.423 m
5 xxx-xxx-internet2 (xx.xx.xx.xx) 52.401 ms 52.340 ms 52.304 ms
6 xx.internet2.edu (xx.xx.xx.xx) 72.170 ms 62.875 ms 62.834 ms
7 xx.internet2.edu xx.xx.xx.xx) 62.783 ms 62.826 ms 62.781 ms
```

Traceroute from San Diego to New York (xxx.xx.xxx.xxx), 30 hops max, 46 byte packets

```
1 xx.xxx.xxx.xx (xx.xxx.xxx.xx) 0.284 ms 0.213 ms 0.236 ms
2 xxx-xxx-xxx xx.xxx (xxx.xx.xxx.xxx) 6.478 ms 6.405 ms 6.381 ms
3 xxx-xxx-x-x xx.xxx (xxx.xxx.xxx.xxx) 6.378 ms 6.318 ms 6.329 ms
4 xxx-xxx-x-x xx.xxx (xxx.xx.xxx.xx) 38.325 ms 42.551 ms 38.378 ms
5 x-x-x-xx.x (x.x.xx.xx) internet2.edu 71.822 ms 61.727 ms 61.841 ms
6 xx-abilene.ucaid.edu (xx.xx.xx.x) 75.263 ms 75.251 ms 75.271 ms
7 xx.internet2.edu (xx.xx.xx.xx) 80.453 ms 80.448 ms 80.461 ms
8 xx.internet2.edu xx.xx.xx.xx) 80.350 ms 80.344 ms 80.359 ms
```

Table 26

10 GigE Network Performance Testing:

The best result in the LAN performance testing has been demonstrated by the combination DELL PowerEdge 1950 - Myricom 10GigE. This architecture has received data at 9.32 Gigabits per second (Table 14 – Graph D). This is indeed a great accomplishment! **Never documented (to the best of my knowledge) in End-to-End testing.**

Obviously, in the WAN testing, performance degradation occurs with an increase of distance and consequently an increase of latency (SD→LOSA=4.04ms, SD→HOUS=36.1ms, SD→WASH=72.9ms). Even with this, I was still able to accomplish several gigabits per second across country.

I consider this:

- a great impact to the 10GigE network performance testing
- a successful corroboration to my settings procedure.

Remarkably, in every test executed, there was a considerable increase in network performances of each system after both tuning and "correct settings" described in detail above. This has been the main focus of this research paper.

Throughout the duration of the entire testing, both in the LAN and WAN, the above combination (DELL-Myricom) has always showed the best performances compared to the couple SUNFire-SUNcard.

The different architecture of the systems and NICs invites a more profound investigation. Although it is not in the scope of this paper, an engineering analysis of the hardware would validate the great result of the DELL 1950 and of the Myricom NIC as a single stream or server-to-server communication tools (iperf 5-parallel-stream option sends to one server and is still considered a single channel). Conversely the examination of the most expensive SUN card and the SUN Fire X4440 (SUN Fire with the 16 CPUs and the SUN card with 20 channels) substantiates the theory that they should perform better in a

distributed (Grid) or multi-streams (servers-to-servers) environment. As I said this could be a really interesting objective for further research.

Packet Capture

Now that we have discovered the true potential throughputs of the 10 GigE versus 1 GigE, let's analyze the second part of the research regarding network packet capture testing.

Packet capture, employed in network intrusion detection systems (IDS), has an entirely different set of requirements than regular NIC operation. The biggest difference is that, in regular NIC operation, the NIC is allowed to (and designed to) drop incoming frames if the host system is unable to keep up or if the frame's destination MAC address doesn't match the NIC's MAC address.

In normal operation, frame (and thus packet) loss is tolerated since the higher level network protocols (such as TCP) can retransmit missing data. In packet capture, the NIC can still drop frames; however, there is no assurance that the information in those lost frames will ever be re-sent. Furthermore, the NIC tries to pass every frame it receives up to the host OS, regardless of whether or not the frame's destination MAC addresses matches the NIC's. This increases the burden on the host and takes time away from reading the NIC's buffers, ultimately resulting in (more) packet and data loss. If there is too much missing data, the network IDS is unable to perform its role.

The network packet capture tests focus on the host and NIC combination to capture traffic with two metrics at extremes: packets per second and payload bytes per second. Packets per second are significant, since for some number of packets received, an interrupt is generated and the host system needs to drop everything and service that interrupt. The more packets per second, the more interrupts per second, and the less time the

host has to do analysis. The second metric, payload bytes per second, indicates how much data per second the host is able to ingest before its internal bottlenecks come into play.

Capture Testing Methodology

To test the packet capture capabilities of the 1GigE and the 10GigE NICs the Spirent AX4000 traffic generator, Force 10 E1200 switch and Bro (open source Intrusion Detection software)[7] have been used (Table 3).

Packet capture performance will be evaluated using the following procedure:

1. Attach host and NIC to test environment.
2. Configure host system and packet generator switch ports to the same non-routable VLAN 10.
3. Verify connectivity. (Do packets from the generator reach the host?)
4. Configure the packet generator to emit packets for the test at various packet sizes (MTU values 576-1500-9000).
5. Run the Bro IDS with a snap length of 9000 and a simple policy for approximately two minutes.
6. Collect the following when Bro is terminated:
   - Exact wall clock run time of Bro (using the "time" utility).
   - Number of packets received by libpcap and successfully passed to Bro.
7. Calculate the rate at which traffic was captured:

   capture rate = (number of packets * packet size) / wall clock run time

8. Compare capture rate with output rate from packet generator, determine the percent loss:

percent loss = 100 * (generator rate - capture rate) / generator rate

To test the maximum number of packets per second a host and NIC combination can capture, the packet generator in the above test is configured to generate random TCP traffic in 576-byte frames. To test the maximum number of bytes per second, the packet generator is configured to emit random TCP traffic in 1500 and 9000 byte frames (not simultaneously).

For testing 1GbE capture, a single output rate of 1Gb/s is used. For testing 10GbE capture, the test at each frame size is repeated at 1Gb/s, 2.5Gb/s, 5Gb/s and 10Gb/s.

The above tests are done using the default kernel and driver tuning parameters, and in cases where further tuning is determined to show an improvement, the tests are repeated with the altered tuning parameters.

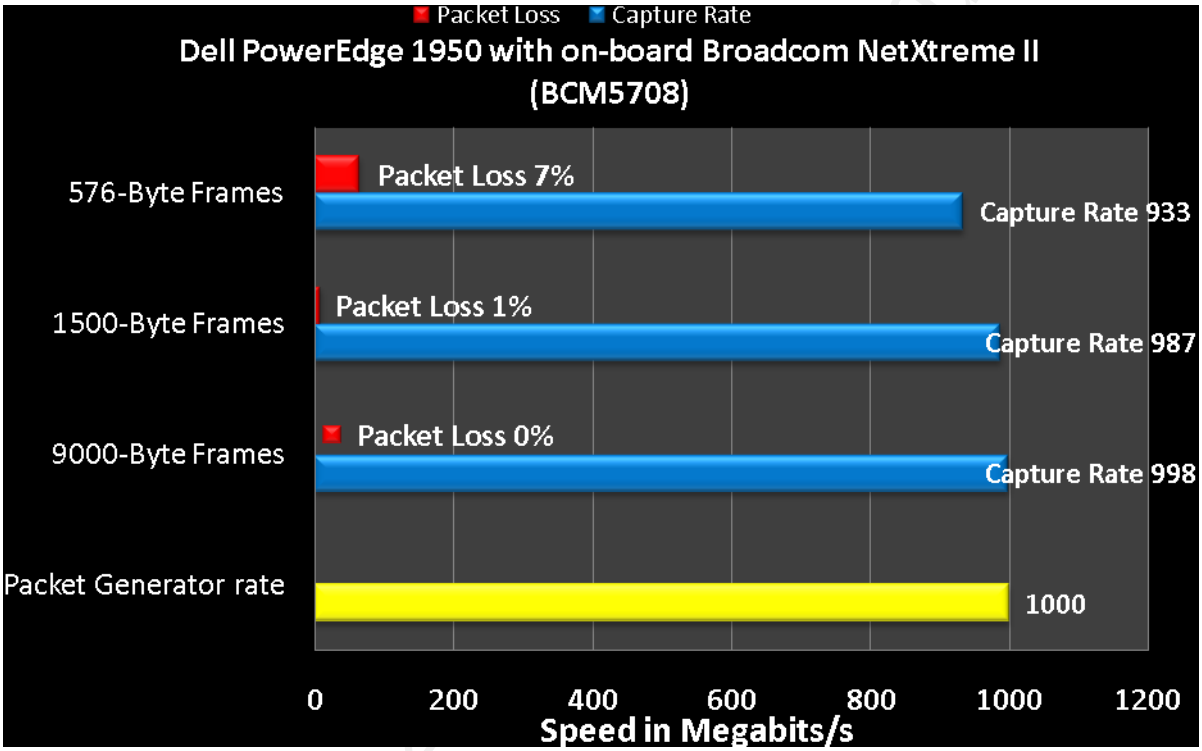
1GbE Capture Test Results

The tables below show the percent loss and capture rate (in megabits per second) for each test. Note that only the results for default tuning are shown. The reason for this will be discussed later.

Dell PowerEdge 1950 with on-board Broadcom NetXtreme II (BCM5708)

Generator Rate	576-Byte Frames		1500-Byte Frames		9000-Byte Frames	
	Packet Loss	Capture Rate	Packet Loss	Capture Rate	Packet Loss	Capture Rate
1 Gbits/s	7%	933 Mbits/s	1%	987 Mbits/s	0%	998 Mbits/s

Table 27

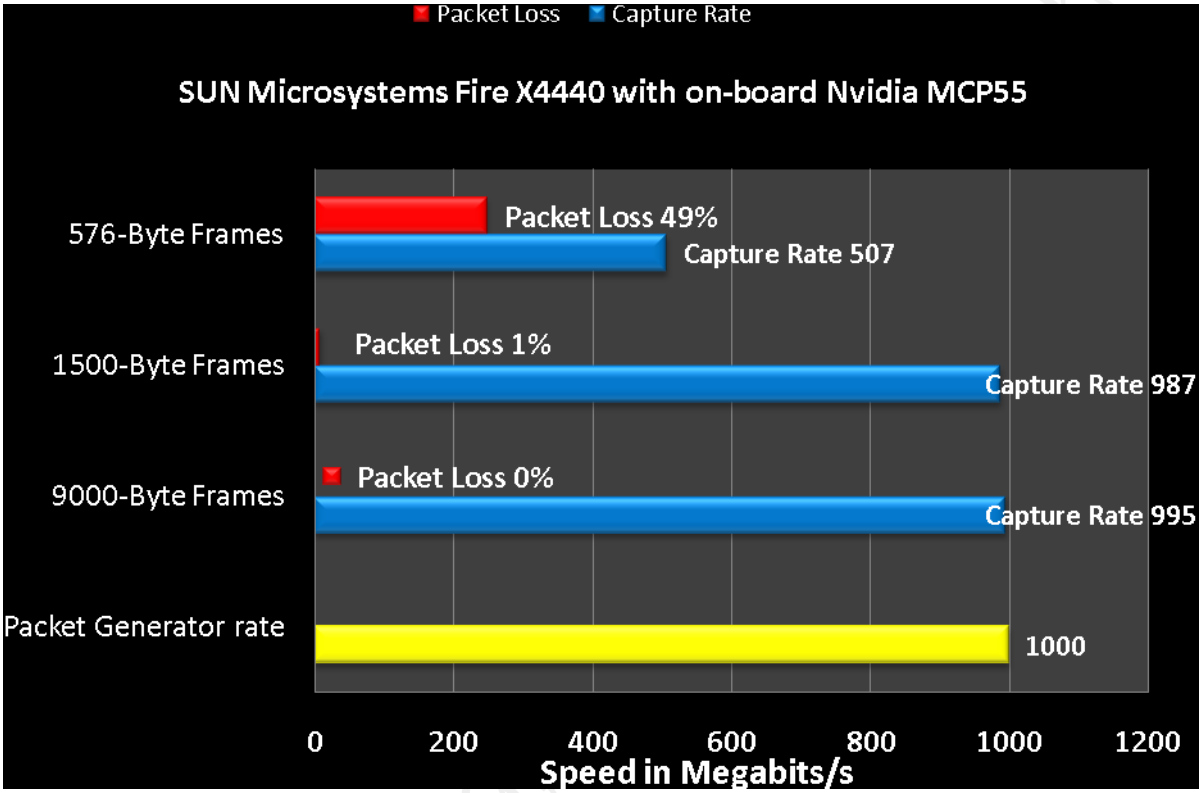


Graph P

SUN Microsystems Fire X4440 with on-board Nvidia MCP55

Generator Rate	576-Byte Frames		1500-Byte Frames		9000-Byte Frames	
	Packet Loss	Capture Rate	Packet Loss	Capture Rate	Packet Loss	Capture Rate
1 Gbits/s	49%	507 Mbits/s	1%	987 Mbits/s	0%	995 Mbits/s

Table 28

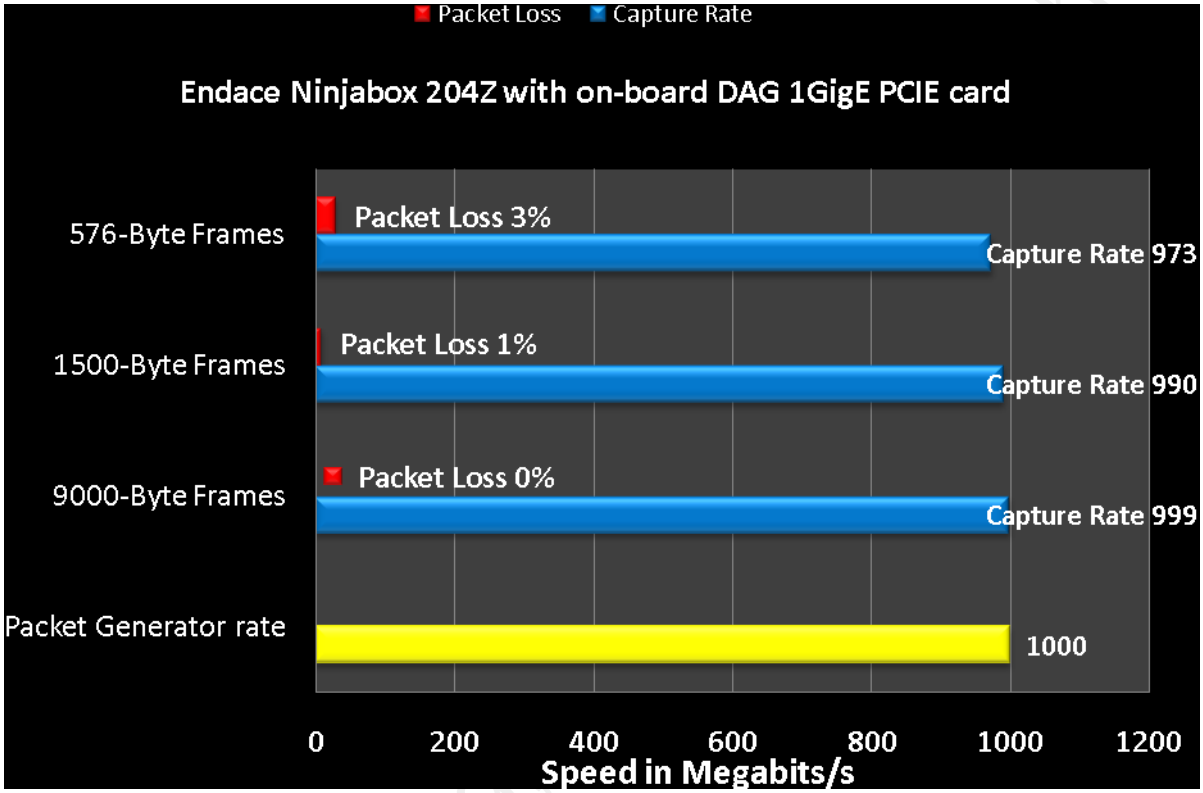


Graph Q

Endace Ninjabox 204Z with on-board DAG 1GigE PCIE card[8]

Generator Rate	576-Byte Frames		1500-Byte Frames		9000-Byte Frames	
	Packet Loss	Capture Rate	Packet Loss	Capture Rate	Packet Loss	Capture Rate
1 Gbits/s	3%	973 Mbits/s	1%	990 Mbits/s	0%	999 Mbits/s

Table 29



Graph R

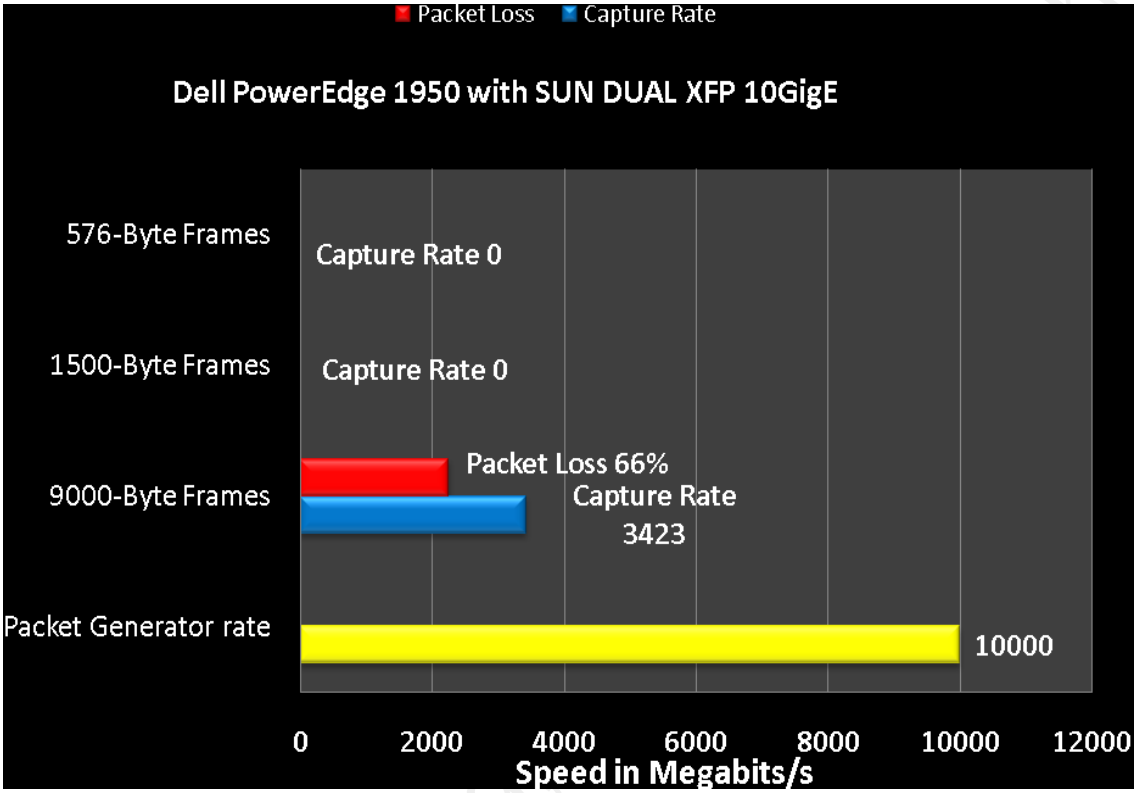
10GbE Capture Test Results

The tables below show the percent loss and capture rate (in megabits per second) for each test. The values in brackets are the values recorded after tuning has been applied.

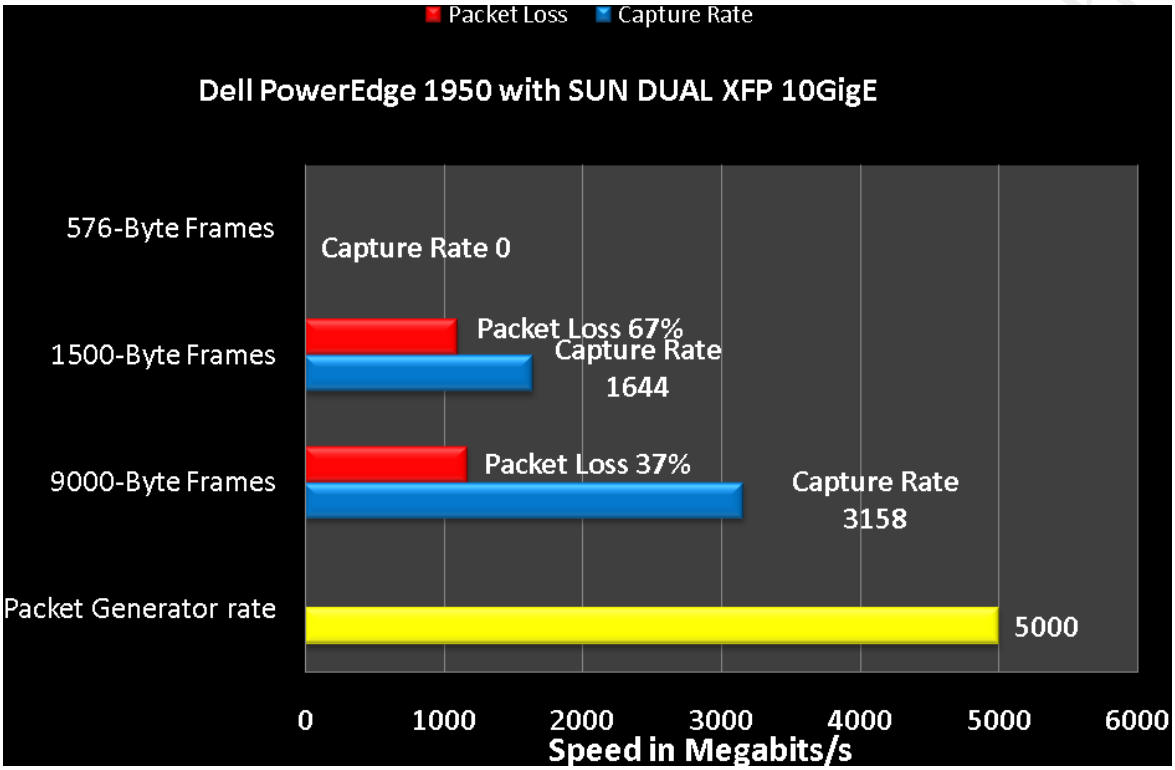
Dell PowerEdge 1950 with SUN DUAL XFP 10GigE

Generator Rate	576-Byte Frames	1500-Byte Frames	9000-Byte Frames
10 Gbits/s	--	--	66%, 3423 Mbits/s
5 Gbits/s	--	67%, 1644 Mbits/s	37%, 3158 Mbits/s
2.5 Gbits/s	66%, 846 Mbits/s	1%, 2467 Mbits/s	0%, 2499 Mbits/s
1 Gbits/s	7%, 935 Mbits/s	0%, 998 Mbits/s	0%, 999 Mbits/s

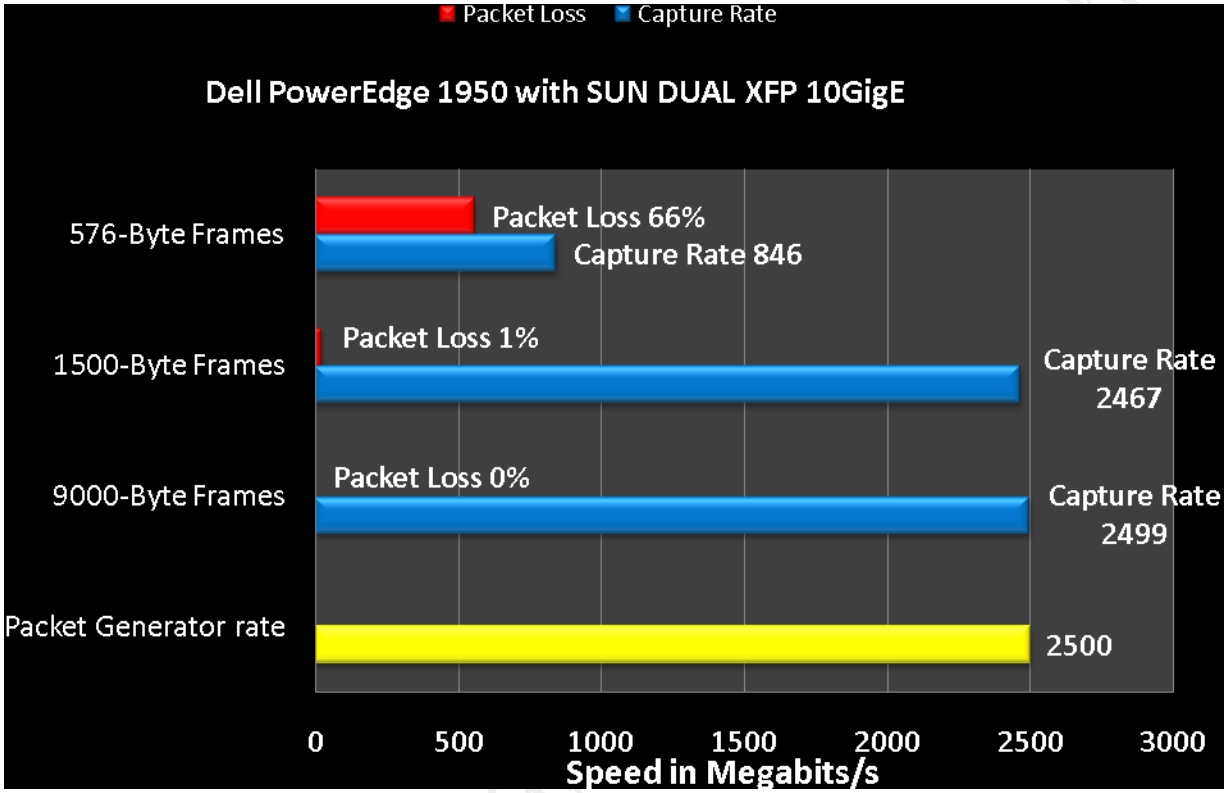
Table 30



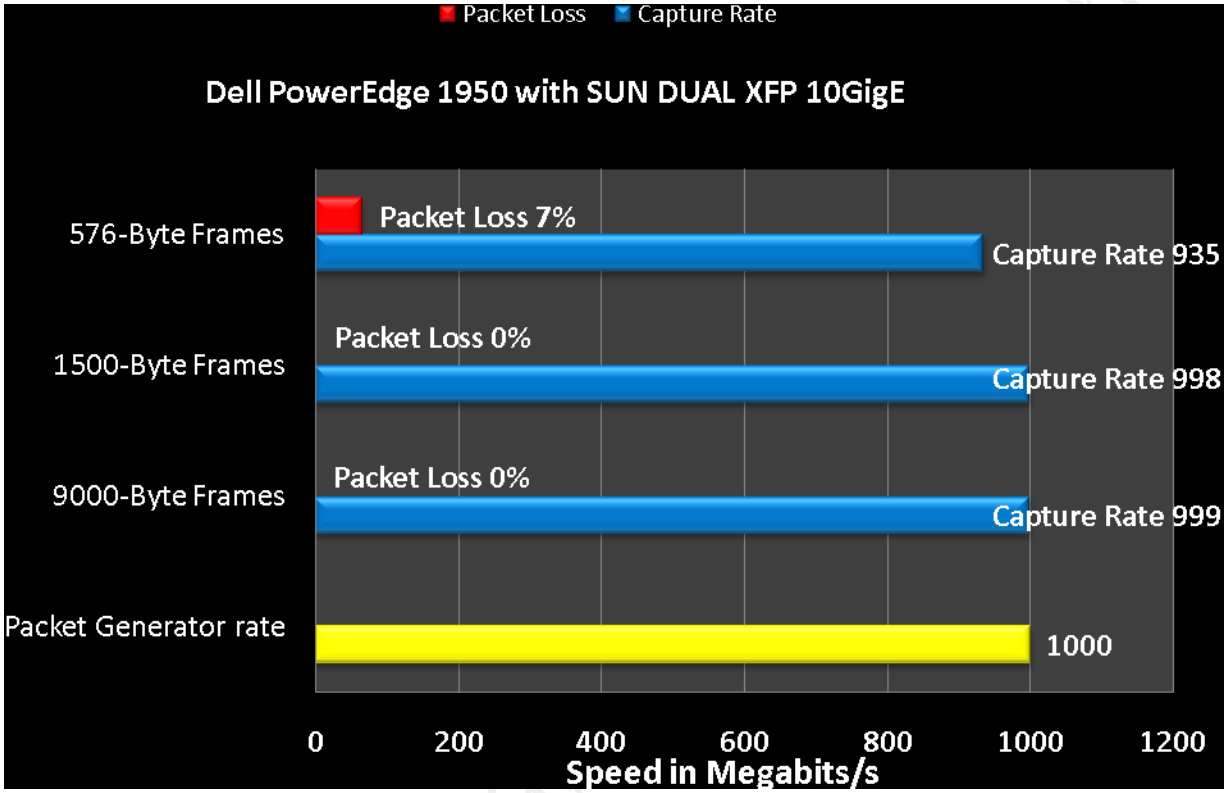
Graph S



Graph T



Graph U



Graph V

Dell PowerEdge 1950 with Myricom 10GigE

Generator Rate	576-Byte Frames	1500-Byte Frames	9000-Byte Frames
10 Gbits/s	94%, 635 Mbits/s [90%, 903 Mbits/s]	87%, 1266 Mbits/s [67%,3277 Mbits/s]	52%, 4881 Mbits/s [32%,6765 Mbits/s]
5 Gbits/s	87%, 647 Mbits/s [83%, 878 Mbits/s]	47%, 2633 Mbits/s [50%,2521 Mbits/s]	24%, 3821 Mbits/s [44%,2803 Mbits/s]
2.5 Gbits/s	70%, 766 Mbits/s [63%, 936 Mbits/s]	18%, 2051 Mbits/s [4%, 2395 Mbits/s]	17%, 2079 Mbits/s [0%, 2499 Mbits/s]
1g Gbits/s	18%, 820 Mbits/s [5%, 950 Mbits/s]	17%, 834 Mbits/s [0%, 998 Mbits/s]	17%, 831 Mbits/s [0%, 999 Mbits/s]

SUN Microsystems Fire X4440 with SUN DUAL XFP 10 GigE

Generator Rate	576-Byte Frames	1500-Byte Frames	9000-Byte Frames
10 Gbits/s	99%, 5 Mbits/s	99%, 16 Mbits/s	59%, 4114 Mbits/s
5 Gbits/s	99%, 5 Mbits/s	87%, 633 Mbits/s	5%, 4770 Mbits/s
2.5 Gbits/s	99%, 5 Mbits/s	24%, 1900 Mbits/s	0%, 2499 Mbits/s
1 Gbits/s	7%, 935 Mbits/s	0%, 999 Mbits/s	0%, 999 Mbits/s

SUN Microsystems Fire X4440 with Myricom 10GigE

Generator Rate	576-Byte Frames	1500-Byte Frames	9000-Byte Frames
10 Gbits/s	91%, 908 Mbits/s [91%, 908 Mbits/s]	80%, 1950 Mbits/s [76%,2420 Mbits/s]	36%, 6394 Mbits/s [45%,5517 Mbits/s]
5 Gbits/s	83%, 858 Mbits/s [83%, 860 Mbits/s]	64%, 1802 Mbits/s [56%,2205 Mbits/s]	3%, 4871 Mbits/s [6%, 4702 Mbits/s]
2.5 Gbits/s	62%, 941 Mbits/s [66%, 828 Mbits/s]	16%, 2106 Mbits/s [2%, 2453 Mbits/s]	0%, 2499 Mbits/s [0%, 2499 Mbits/s]
1 Gbits/s	43%, 567 Mbits/s [3%, 974 Mbits/s]	0%, 999 Mbits/s 0%, 999 Mbits/s	0%, 995 Mbits/s [0%, 999 Mbits/s]

IBM eServer x345 with Endace DAG 6.2SE*

Generator Rate	576-Byte Frames	1500-Byte Frames	9000-Byte Frames
10 Gbits/s	80%, 1967 Mbits/s	63%, 3726 Mbits/s	45%, 5474 Mbits/s
5 Gbits/s	61%, 1945 Mbits/s	18%, 4096 Mbits/s	0%, 4983 Mbits/s
2.5 Gbits/s	23%, 1932 Mbits/s	0%, 2488 Mbits/s	0%, 2491 Mbits/s
1 Gbits/s	0%, 998 Mbits/s	0%, 997 Mbits/s	0%, 997 Mbits/s

Analysis of Packet Capture Test Results

1 GigE:

The systems that I tested for gigabit packet capture appear to be capable of full-speed capture with no additional tuning required. Furthermore, no further tuning was able to improve upon the results obtained with a default configuration. For that reason, there were no improved results to show.

I did discover, however, that CPU binding has an effect. Binding the IDS process to the same CPU servicing the interrupt of the NIC doing the packet capture will decrease performance, in some cases, to a small fraction of full line speed. It stands to reason, then, that packet capture and IDS should not be performed using single-cpu systems.

The packet capture tests were repeated using an Endace Ninjabox, a packet capture appliance, and the results were included to show how specialized hardware performed on the same set of tests. As can be seen, both the Endace hardware and commodity hardware were able to do line-speed packet capture. It is worth noting, however, that the Endace equipment utilizes DMA transfers into large buffers in system RAM, which may permit sustained line-speed capture rates under higher CPU load (such as when running more complex IDS analysis).

10 GigE:

The demands placed upon a system doing 10-gigabit packet capture (and regular network throughput) are much higher than one doing 1-gigabit packet capture, especially on interrupt load. This extra burden became very evident when testing the Sun 10GE card

in the Dell 1950, which actually caused the system to become unresponsive when placed in promiscuous mode while it was receiving 576-byte and 1500-byte frames at 10gb/s, as well as when it was receiving 576-byte frames at 5gb/s.

Manufacturers have different techniques to achieve high throughput with their NICs and the difference in those techniques become evident when their NICs are used for packet capture. The SUN 10GigE NIC, in particular, did not seem to yield improved packet capture performance when its driver parameters or kernel core networking parameters were changed. That is why there are no improved results to show.

The Myricom card responded nicely. I used the following commands to achieve a higher packet capture rate on both the Dell and SUN host:

```
ethtool -K eth5 tx off tx off tso off sg off
ethtool -A eth5 autoneg off tx off rx off
ethtool -C eth5 rx-usecs 0
sysctl net.core.rmem_max=73400320
sysctl net.core.rmem_default=73400320
sysctl net.core.netdev_max_backlog=1000
```

The only consistent tuning factor between the two cards is the selection of CPU core to run the IDS process on. Much like 1 GigE packet capture, binding to a CPU core that isn't handling the NIC's interrupt is important to getting the highest performance. Unlike network throughput, the selection of core with respect to L2 cache does not seem to matter, as long as the core running the IDS process is not the same one running the NIC. In most cases, the kernel won't schedule the IDS process on the same CPU core that is handling the NIC, though it's not a bad idea to make sure that doesn't happen at all. The penalty for sharing the core

between the NIC and IDS process is much greater in this case and can drop capture performance to 1/100th of "normal" speed.

For reference, an Endace DAG 6.2SE card was tested in an older IBM eServer x345. It's worth noting that the DAG card runs on a PCI-X bus (133 MHz, 64-bit) on a somewhat slower system when compared to the above 10GbE NICs, which are using a 4-lane PCI-E, interface to modern server-grade hardware. Yet, even with a slower CPU and more restrictive host communication path, the DAG card was able to capture more traffic than either of the two NICs (except with 9000-byte frames at 10 Gigabits/s), especially with 576-byte frames.

The higher DAG card performance suggests that the bottleneck in the Sun and Dell system and NIC combinations lies in an upper limit to the number of interrupts per second they can handle. Indeed, in the case with the Sun 10GE NIC, tuning the driver to generate interrupts more frequently reduced its capture performance. Attempts to increase interrupt coalescing from default did not yield increased capture performance, but that may be due to hitting some hard-coded value limit within the driver. Modifying both the Myricom and Sun driver for improved capture performance may be the subject of further research.

Recommendations Based on Results

For 1GigE network performances and capturing, there are a few well-known tuning settings to achieve better performance. For 10GigE, there are many applicable settings involved in tuning the kernel and the 10GigE cards. To complicate the situation, those settings may vary based on the card brand and system architecture. Therefore, one "catch-all" recommendation is not the one I can advocate.

The physical ID and the CPU interrupt binding (NIC or other processes) CHANGE every time the system reboots. My advice is to proceed checking them every time the need to transfer large data or perform a network testing arises.

As stated above before doing any check on the CPU binding, make sure to get the chipset architecture in hand and also experiment with it to verify which method is the best for your brand since the performances and the methodology of the CPU binding change based on the Brand and architecture of the system.

The DELL1950 demonstrated poor performances when a core used by the application (iperf) was on the same CPU that handled the NIC. The same applied if a different core on the different CPU is used.

The only time the highest numbers are obtained, in term of performances, is when I bound the application to another core in the same CPU (sharing the same L2 cache).

The reason is that the application and the NIC do not have to go out to I/O to get memory to run but they share the same memory (saving time and bandwidth).

Conversely, I discovered that for the SUN system architecture, the Sun Fire X4440, the best performances were obtained when I

bind the application to the same core that handles the NIC interrupt (using the same phisicalID for both). The reason is that the SUN has an L2 cache memory for each core not shared.

1GigE and 10GigE Packet Capture:

Packet capture may be at odds with the goal of high network performance. High network performance favors low latency, which implies smaller buffers and more interrupts.

Packet capture performance, on the other hand, is improved with fewer interrupts (which requires larger buffers). The test results suggest that 1-gigabit capture can be performed with NICs up to line speed.

However, 10-gigabit capture using a regular NIC seems to lose its effectiveness at speeds over 2.5 Gbits/s.

If packet capture on a 10-gigabit link needs to be performed with a NIC, evaluating different hardware is absolutely necessary, because different brand and model NICs can have widely different results.

Above all, a host architecture employing multiple CPUs (not a Hyper-Threaded CPU) is a necessity.

References

1 http://www.psc.edu/networking/projects/tcptune/ (2008) Internet

2 http://www.internet2.edu/pubs/networkmap.pdf (2008) Internet

3 http://dast.nlanr.net/Projects/Iperf/ (2008) Internet

4 http://docs.sun.com/source/820-1606-11/LP_driverparams.html#50450458_91890 (2008) Internet

5 http://www.hep.ucl.ac.uk/~ytl/mb-ng/wan-txq-tcp/index.html (2008) Internet

6 http://www.myri.com/serve/cache/511.html#linux (2008) Internet

7 http://www.bro-ids.org/ (2008) Internet

8 http://www.endace.com/ninjabox.html (2008) Internet