Performance Evaluation of Linux Bridge

James T. Yu

School of Computer Science, Telecommunications, and Information System (CTI)

DePaul University

ABSTRACT

This paper studies a unique network feature, Ethernet bridge, in the Linux kernel and conducts an extensive experiment to measure its performance as defined in RFC-2544. The result shows that Linux bridge yields *satisfactory* performance if the system occupancy is less than 56% where its latency and throughput are comparable to Cisco Catalyst 2950 switch. The performance is considered *acceptable* until system occupancy reaches 85%. We also compared the performance between Linux bridge and Linux router, and the results are almost the same as measured by latency and throughput.

The contributions of this study are summarized as follows:

1. With its open source, Linux bridge is like *programmable* switch for education and research. We are encouraged by the performance results of this study, and plan for more advanced research on Linux bridge in load balancing and high availability.
2. The performance result shows that Linux can be deployed as an effective network device if its occupancy is properly engineered for targeted applications. One example is network *firewall* where the Wide Area Network (WAN) link is usually than 10M.
3. Our experiment of bridge and router configuration can be used for classroom demo and lab exercise on data network education. The use of RFC-2544 serves as a useful guide to learn network benchmark testing and performance measurements.

1. Introduction

Linux, since its introduction by Linus Torvald in 1991, is well received by the industry and academics for various business, personal, and research applications. It replaced Mac OS as the number 2 desktop OS in 2002, and ran on 25% corporate servers in the same year [1]. In addition to business and end-user applications, Linux has a unique network feature, Ethernet bridge, which is relatively new to many people. This feature is in the Linux kernel and conforms to the IEEE 802.3 standard [2][3]. Windows and UNIX do not support this feature of Ethernet bridge. The purpose of this paper is to study the performance of Linux bridge and explores its applications for industry and academic uses.

2. Linux Bridge Configuration

In general, any workstation with two network interface cards (NIC) can be configured as a router. These two NICs require two different IP addresses on two separate IP subnets as illustrated **Figure 1**.

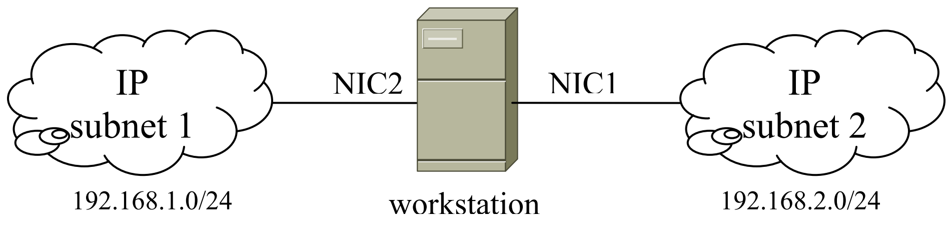

Figure 1. Workstation Configuration for IP Router

In this configuration, we can simply turn on the *IP forwarding* capability and the workstation becomes an IP router. The procedure on Linux for this configuration is one simple command:

```
echo "1" > /proc/sys/net/ipv4/ip_forward
```

The configuration procedure for IP routing on the Windows (NT, 2000, and XP) and Unix is similar to Linux. The protocol stacks for the IP router configuration are given in Figure 2.

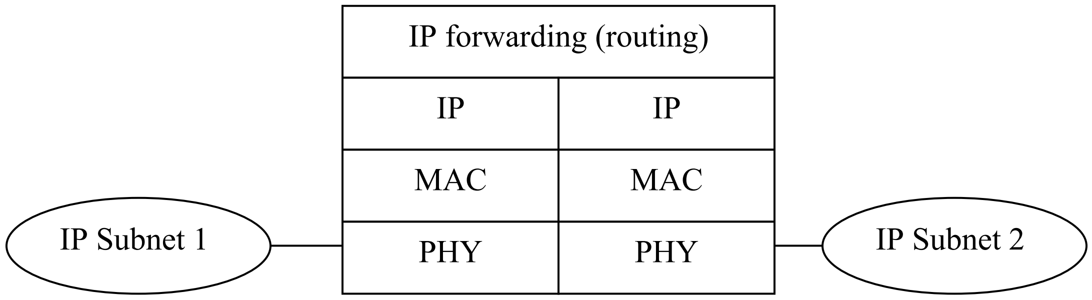

Figure 2. Protocol Stacks for IP Routing

On the other hand, it is more complex to support *bridge* configuration where multiple physical interfaces (or NICs) share a single IP address. The bridge configuration is the foundation for many firewall products, and it usually requires expensive software for it. The Linux operating system has the bridge software in its kernel and this feature can be turned on by the bridge utility[5]. In this configuration, a Linux workstation has multiple physical interfaces (NICs) but only one IP address as illustrated in Figure 3.

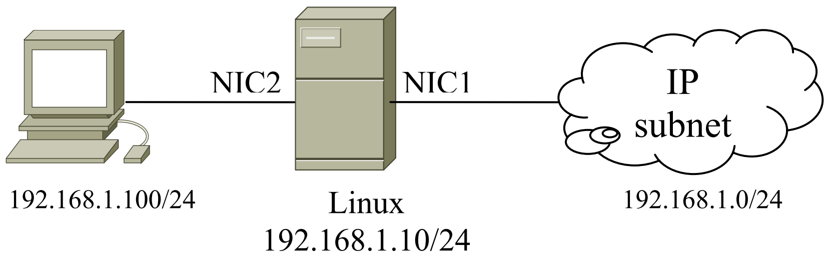

Figure 3. Linux Configuration for Ethernet Bridge

In order to create an Ethernet bridge interface, we need to add a bridge layer between the IP layer and the Ethernet (MAC) layer as illustrated in Figure 4.

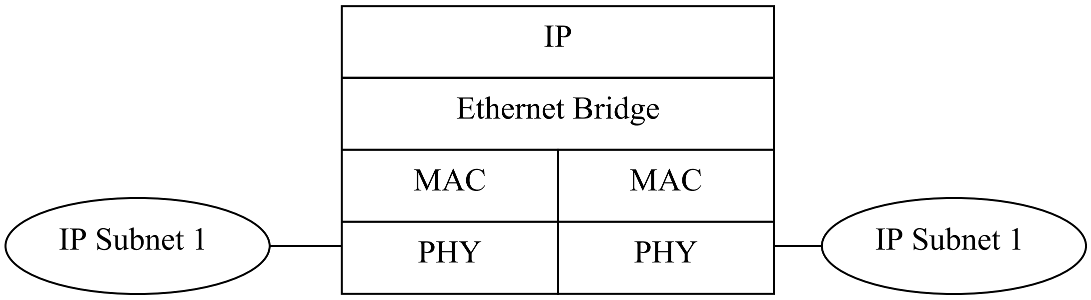

Figure 4. Protocol Stacks for Linux Ethernet Bridge

By default, an IP address is bound to a physical interface. To create a bridge interface, we need to remove the binding of the original IP address to the physical interface and then create a new IP address for the new bridge interface. An example of the configuration procedure is given in the following script.

```
# ************ Create a bridge interface and it is called br1
brctl addbr br1

# ************ Add physical interfaces to the bridge interface
brctl addif br1 eth0
brctl addif br1 eth1

# ************ Reset IP interface
ifconfig eth0 0.0.0.0
ifconfig eth1 0.0.0.0

#Bring up the bridge
ifconfig br1 up

# **********  Set IP address of the bridge
ifconfig br1 192.168.1.10 netmask 255.255.255.0 up

# **********  Set IP default gateway
route add default gw 192.168.10.1
```

This bridge configuration also supports the Spanning Tree Algorithm and Protocol (STP) as defined in IEEE 802.1D [4]. The objectives of this research are to study the performance measurements of Linux bridge and explore its applications for business and academic uses.

3. Performance Measurement Environment

We follow the standard of RFC-2544 for benchmark performance testing [6] where the key measurements are frame loss, **throughput** and **latency** as defined in RFC-1242 [7]. The cause of frame loss on Ethernet is primarily due to *congestion* which causes buffer overflow. In this case, we shall also notice longer latency and poorer throughput. Therefore, we do not present the data of frame loss as it is implied from the other two measurements in our experiment. We applied two control variables for the performance test: *input frame rate* (frames per second or fps) and *frame size*. Another important measure is Linux *system occupancy* which is the percentage of CPU time consumed by the running processes.

The school of CTI at DePaul University received an equipment grant from IXIA, Inc. and the grant allows us to acquire the IXIA traffic generator and analyzer. This equipment provides the opportunity to conduct advanced and high performance network measurements. Our first baseline measurement is on the IXIA box itself, and its physical connection is illustrated in Figure 5 which is a loop-back connection.

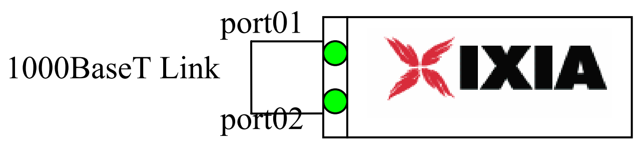

Figure 5 IXIA Configuration for Baseline Performance Test

Port-01 of the IXIA box is used for data transmission and port-02 is used for reception. Each data frame has a time stamp which is used to calculate *latency* at the receiving port. The performance data is collected and presented in **Table 1**.

Table 1. Baseline Measurement of IXIA Traffic Generator

Traffic Generation	Frame Size (in bytes)	Throughput (in bps)	Latency (µs)
1 frame per second	64	512	1.8 µs
1G bps	64	760M	1.8 µs
1 frame per second	1500	12,000	1.8 µs
1G bps	1500	1G	1.8 µs

We have the following observations on this baseline testing :

- The data shows that the IXIA box is a *wire-speed* traffic generator and capturer where its performance is almost the same as the physical cable with zero frame loss. The latency is a constant value even at the max input rate.
- The latency is independent of the packet size because the latency measurement, as defined in the standard[7], is "the time interval starting when the last bit of the input frame reaches the input port and ending when the first bit of the output frame is seen on the output port."
- The throughput for the minimal frame size is the same as the theoretical limit.

Inter Frame Gap (IFG) = 96 bit timer = 96/1,000M = 0.096 µs

Preamble = 48 bit timer (8 bytes) = 8/1,000M = 0.064 µs

Slot time = minimum frame size / 1,000M = 512/1,000M = 0.512 µs

Link utilization = 0.512 ÷ (0.512 + 0.096 + 0.064) = 76%

Throughput (theoretical limit for min size frames) = 760M bps

- We observed the throughput close to 1G bps by using the max frame size.

The second experiment is to baseline the performance measurements of two Cisco Catalyst 2950 switches which are required for Linux performance testing later. The physical connection for the test is illustrated in Figure 6.

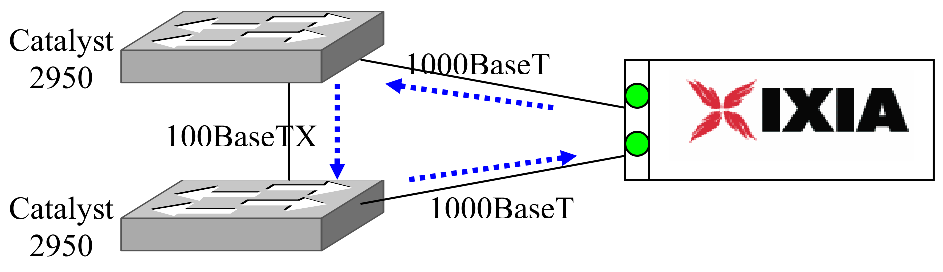

Figure 6 Catalyst 2950 Switch Configuration for Baseline Test

The test results of latency and throughput are shown in Figure 7 and Figure 8. The frame input rates for the latency test are relatively low to avoid congestion.

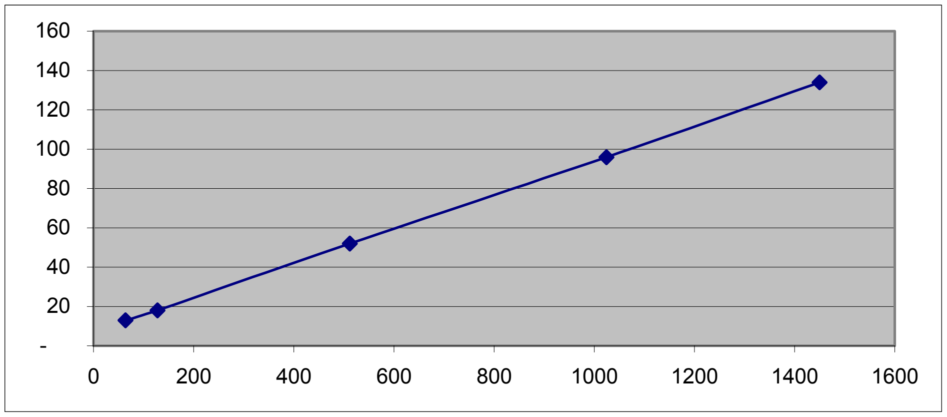

Figure 7. Catalyst 2950 Latency Test (Latency in μs vs. Frame Size)

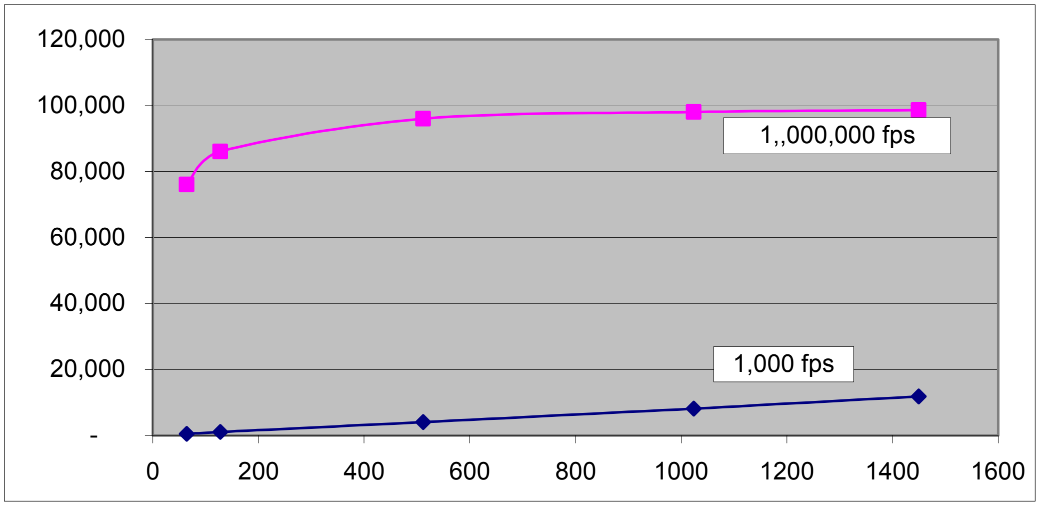

Figure 8. Catalyst 2950 Throughput Test (TP in Kbps vs. Frame Size)

We also run an overload test for the min frame size with input rate greater than the theoretical limit (76M bps). The result is given in Table 2.

Table 2. Catalyst 2950 Overload Test (Frame Size=64 bytes)

FPS	XMIT Rate (bps)	Latency (us)	Throughput (bps)
10	5,120	13.5	5,089
10,000	5,120,000	13.5	5,100,000
100,000	51,200,000	13.5	48,000,000
160,000	81,920,000	13.5	75,000,000
161,290	82,580,645	13.5	75,600,000
162,602	83,252,033	100.0	76,000,000
163,934	83,934,426	8,300.0	76,000,000
1,000,000	512,000,000	8,100.0	76,000,000

We have the following observations:

- If the network is not congested, latency is independent of the frame input rate.
- When the physical link is congested as shown in Table 2, latency increases by 2-3 orders of magnitude.
- Catalyst 2950 switch uses the *store-and-forward* method for frame forwarding, and the observed latency is close to the theoretical limit. For example, for the frame size of 1,500 bytes, the theoretical limit is 120 µs, and our observed value is 126 µs. Note that the latency measure is on the switch and its 100BaseTX link, excluding the delay of IXIA box and the 1000BaseT link.
- For minimal frame size, we observed a throughput equal to the theoretical limit of 76Mbps. With the max frame size (1,500 bytes), we observed a throughput close to 100Mbps.

Our conclusion is that Catalyst 2950 has near *wire-speed* performance. In our later testing with Linux bridge, both Catalyst switches would not cause any congestion issues as long as the input rate is less than the theoretical limit (76M for 64-byte frames and 99M for 1,500-byte frames).

4. Linux Bridge Performance Test

As discussed in Section 2, a Linux machine with two or more interfaces can be configured as an Ethernet bridge. After we tested and validated the Linux bridge configuration as illustrated in Figure 3, we connected the Linux bridge to the IXIA traffic generator as illustrated in Figure 9. The reason we need two Catalyst switches is to convert 1000BaseT traffic to 100BaseTX traffic to the Linux bridge which does not have the 1000BaseT NIC. The traffic generator, on the other hand, supports 1000BaseT only. This configuration also supports overload tests with various congestion scenarios. As we have demonstrated in Section 3, the Catalyst switch would not cause any congestion issue as long as the input rate is less than the theoretical limit.

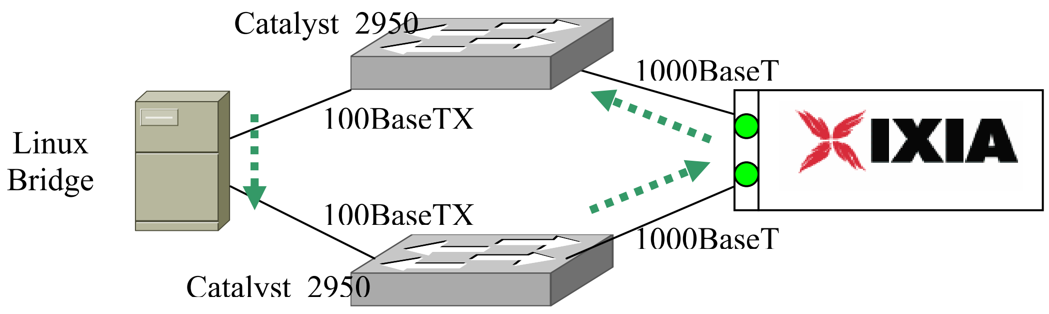

Figure 9. Linux Bridge Test Configuration

We first conducted the latency test where we set different frame sizes and frame input rates on IXIA port01. The latency test results are illustrated in Figure 10 where we include the data of Catalyst 2950 switch for comparison.

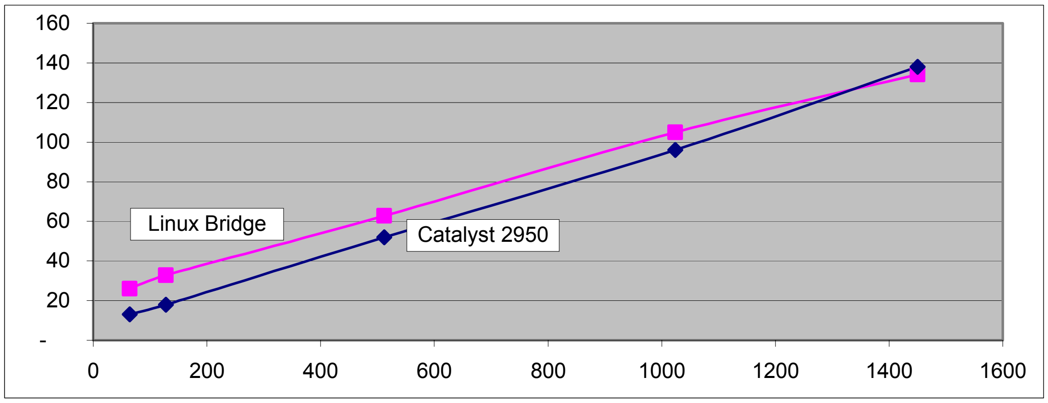

Figure 10. Linux Bridge Latency Test (us vs. Frame Size) – nominal input rate

It is interesting to see that the latency of Linux bridge is comparable to Catalyst 2950 switch at the nominal input rate (1,000 fps), and this latency is close to the theoretical limit of Ethernet.[1] One interesting observation is that when the input rate is low (1 fps), the latency of Linux bridge is longer than that of higher input rate (1,000 fps). This result is due to the characteristics of the Linux operating system where *tasks* are scheduled at a fixed interval. When a frame arrives but the *bridge* task is not scheduled, this frame will experience longer delay. This small variation (up to 50 µs) should not be an issue to most network applications.

Our next test is to conduct an *overload* test and observe the Linux bridge behavior. We turned on the performance monitor on the Linux bridge:

sar –u 5 100   # show Linux CPU occupancy every 5 seconds for 100 runs

The CPU of the Linux machine is AMD Duron 1.3GHz with 512M memory. The overload test results are shown in **Table 3.**

[1] The comparison of Linux bridge with Catalyst 2950 is on a single 100BaseTX port. A Catalyst 2950 switch (2950T) has 24 100BaseTX ports and 2 1000BaseT ports, plus a switch fabric. Its aggregated capacity is probably 100 times more than a Linux bridge.

Table 3. Linux Bridge Latency Overload Test (64 bytes)

FPS	Input Rate (bps)	Latency (us)	Throughput (bps)	Linux CPU Occupancy
100	51,200	25	52,000	0%
1,000	512,000	25	511,000	2%
10,000	5,120,000	25	5,000,000	21%
20,000	10,240,000	25	10,000,000	42%
40,000	20,480,000	25	20,000,000	56%
50,000	25,600,000	29	24,000,000	70%
55,556	28,444,444	29	28,000,000	77%
58,824	30,117,647	29	29,000,000	82%
62,500	32,000,000	32	30,000,000	85%
64,516	33,032,258	36	32,000,000	85%
65,574	33,573,770	37	32,300,000	86%
66,225	33,907,285	87	32,700,000	N/A*
66,667	34,133,333	22,987	25,000,000	N/A*
100,000	51,200,000	44,987	18,000,000	N/A*

*All user applications (including **sar**) are blocked for a lack of CPU resource.

The overload test shows that the CPU is the *limiting* factor for the overall bridge performance. Linux bridge shows the performance comparable to Catalyst 2950 (a commercial switch) when the occupancy is less than 56% where the machine can process up to 40,000 frames per second (fps). We observed *some* performance degradation when the occupancy increases from 56% to 85%. The congestion occurs as the occupancy reaches 85% where the latency jumps from 87μs to 23ms. The throughput also goes down from 33M to 25M bps. If input rate continues increasing, the perform keeps going down for both latency and throughput.

The last test is to measure the throughput and the results are shown in Figure 11 where we observe a throughput of 96M bps with the max frame size (1,500 bytes) and an input rate greater than 100M bps. Although the latency is very high (tens of milliseconds), we are glad to see a throughput close to the physical limitation.

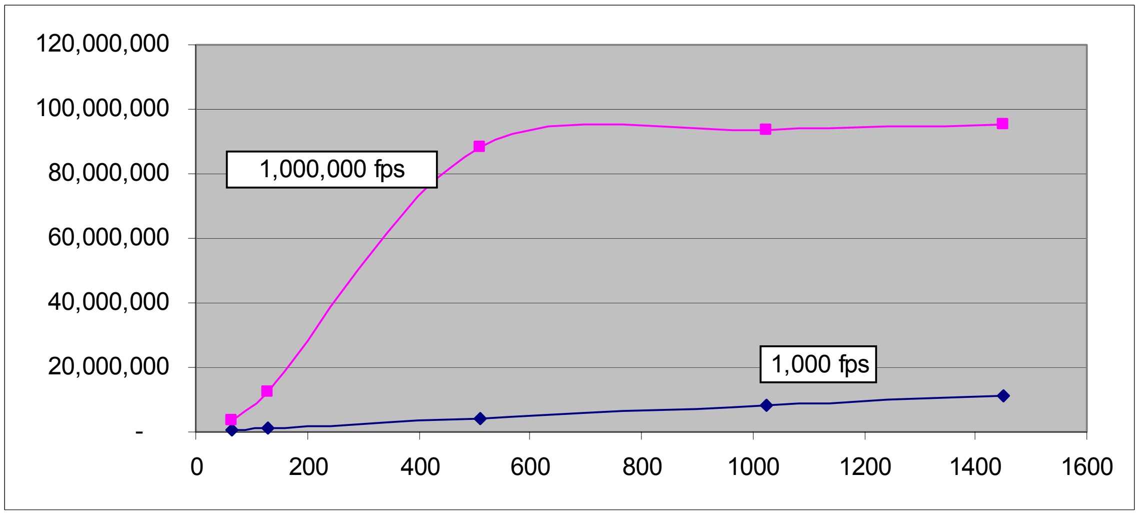

Figure 11. Linux Bridge Throughput Test (TP in bps vs. Frame Size)

5. Linux Router Performance Test

In addition to the Linux bridge performance, we are interested in the Linux routing performance where we expect the Layer-2 performance (bridge) to be significantly better than the Layer-3 performance (router). We first successfully tested the router functionality as illustrated in Figure 1. The physical connection for the Linux router performance test is illustrated in Figure 12 where it also shows the IP address scheme for the routing test.

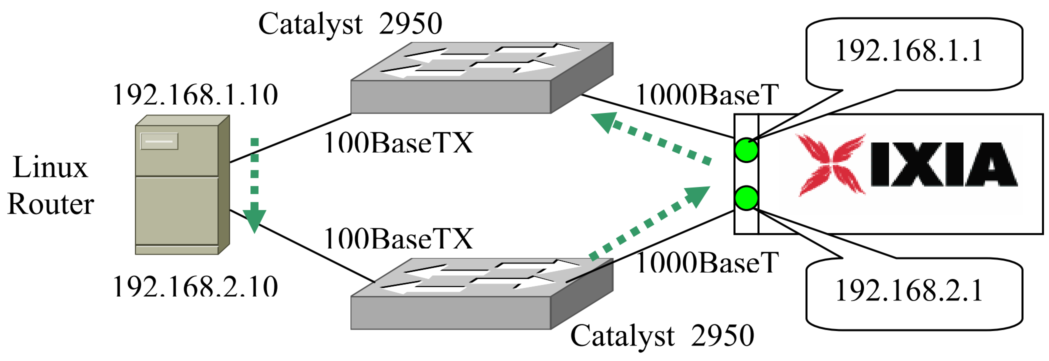

Figure 12. Linux Router Performance Test Configuration

We conducted the same latency and throughput tests and the results are illustrated in Figure 13 and Figure 14. The latency test was conducted at a low input rate (1,000 fps), and we do not observe much variation with input rates as long as the system occupancy is within 56%.

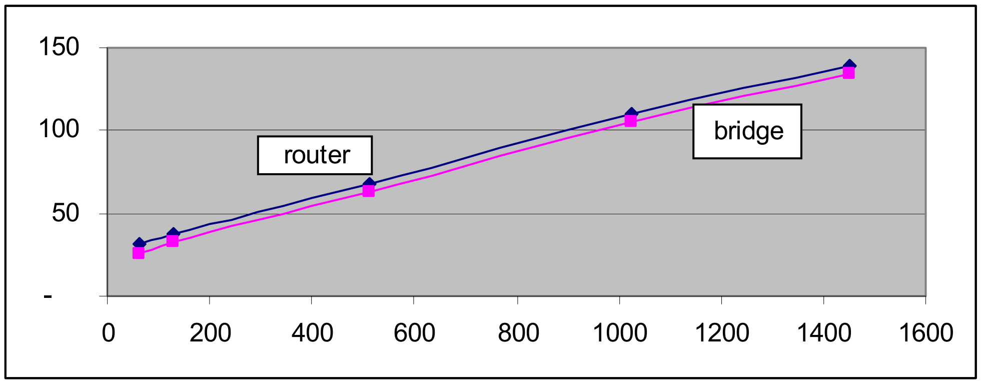

Figure 13. Linux Router Latency Test (Latency in μs vs. Frame Size)

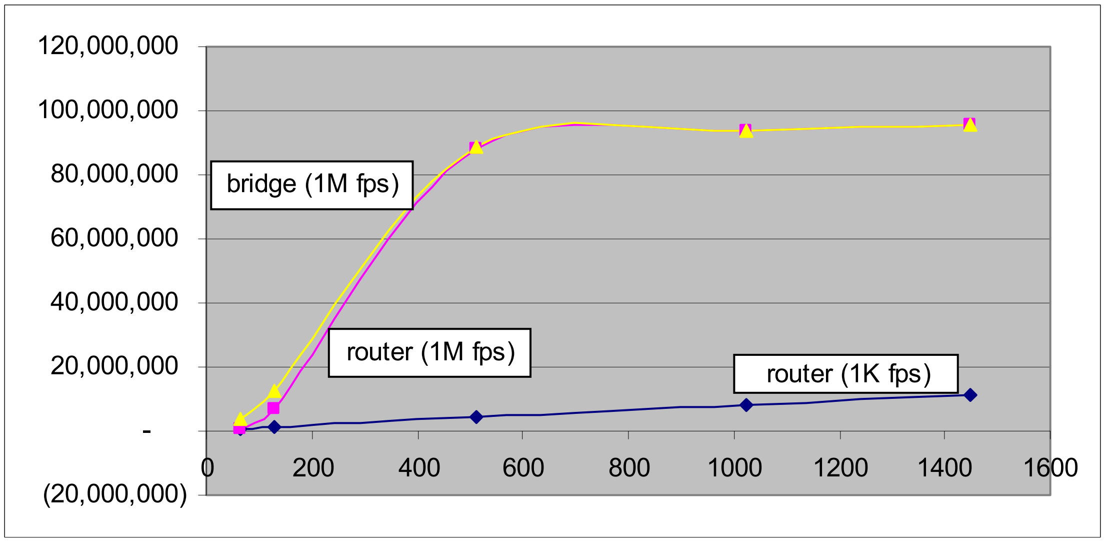

Figure 14. Linux Router Throughput Test (TP in bps vs. Frame Size)

The data of Linux bridge is added into both figures for comparison where we observed 5 μs longer latency for the Linux router, and this difference is independent of the frame size. Our interpretation is that it takes a little more CPU resource to read an IP packet while the process of IP routing table and MAC forwarding table takes about the same time

The throughput of Linux router is similar to that of Linux bridge (Figure 11) except for small frames where bridge yields better performance than router (3M bps vs. 0.6M bps for 64-byte frames). When the frame size increases to 512 bytes, we do not observe any difference in throughput between bridge and router. When the input rate is low (1,000 fps), the throughput is identical between bridge and router, regardless of frame sizes. As we observed in the latency data, this result is consistent with our understanding of how Linux router works. (The process of routing table and MAC forwarding table consumes same amount of resource.) As we noted earlier, we can push the routing throughput close to the physical limit by using the max frame size and a high input rate where we observed 96M bps.

6. Conclusions and Future Work

We conducted a detailed study of the Linux bridge performance based on the standard benchmark testing as defined in RFC-2544. The performance data of Linux bridge is compared to that of a commercial Ethernet switch (Catalyst 2950) on a single port basis, and the results are comparable if the Linux CPU occupancy is below 56%. In this case, the performance is close to the theoretical limit. We observed slightly performance degradation when the CPU occupancy increases from 56% to 85%. After that, congestion occurs and the performance goes down by 2-3 order of magnitude. Because the Linux CPU of this study is only 1.3GHz, we are interested in further study with more powerful processors. In addition, we expect to see higher performance with server NIC which has the processing power on the NIC.

We also compared the performance between Linux bridge and Linux router. In contrary to our initial expectation, the results are almost identical for latency and throughput. Our interpretation is that the process of the IP routing table and the MAC forwarding table takes about the same amount of CPU resource.

The conclusion of the study is that Linux can be served as an effective platform for network applications and research. If one can engineer the CPU resource with 56% reserved for the network layer, the user applications would experience satisfactory results. One application is network firewall where the Wide Area Network (WAN) link is usually less than 10M. A Linux-based firewall would have no performance issue for such an application [8].

Another contribution of this study is to use Linux for data networking education. We provided detailed procedure for the experiment and it can be demonstrated in a classroom setting or as a lab exercise. An instructor can configure Linux for teaching Ethernet in one week, and configure the same box for IP routing the following week. The RFC-2544 standard is a useful guide for learning benchmark testing and performance measurements.

Lastly, Linux provides the source code of Ethernet bridge, and we can modify the source code for advanced research on data networking. For example, we are looking into two new Ethernet standards, Rapid STP (IEEE 802.1w) [9] and Link Aggregation (IEEE 802.3ad) in the areas of load balancing and fault tolerance. This research requires enhancement to the existing source code of Linux kernel For the purpose of education, we can turn on the trace in the source code, and Linux can show the details of the MAC learning, STP state transition, various bridge timers, and the content of Bridge Protocol

Data Unit (BPDU). We hope this paper will bring more interest in Linux bridge and moiré support of enhancement to its source code.

References

[1] "The Big Guys Latch Onto Linux," *BusinessWeek*, March 2003.

[2] Source code of Ethernet bridge is at /usr/src/<LinuxVersion>/net./bridge.

[3] IEEE 820.3, CSMA/CD Access Method and Physical Layer Specifications

[4] IEEE 802.1D, Media Access Control (MAC) Bridges.

[5] http://www.tldp.org/HOWTO/BRIDGE-STP-HOWTO/

[6] RFC-2544. Benchmarking Methodology for Network Interconnect Devices.

[7] RFC-1242. Benchmarking Terminology for Network Interconnection Devices

[8] http://www.grennan.com/Firewall-HOWTO.html

[9] J. T. Yu, "Applying IEEE 802.1w (RSTP) to Improve Service Availability," *IEEE International Conference on Dependable Systems and Networks,* (June 2003), pp. B10-B11.