Evaluation of the FreeBSD dummynet network performance simulation tool on a Pentium 4-based Ethernet Bridge

W.A Vanhonacker
Centre for Advanced Internet Architectures. Technical Report 031202A
Swinburne University of Technology
Melbourne, Australia
wendy.vanhonacker@epfl.ch

***Abstract*-This technical report describes a simple procedure to estimate dummynet performance. Dummynet is a module installed on the bridge that will allow us to subject selected packets to delays, random loss or bandwidth limitation. The goal is to verify how precise is dummynet since in a future project we might use it as a traffic shaper for a network simulation. This report is mainly based on another report from A.L.Cricenti: "Evaluation of a Pentium PC for use as an Ethernet Bridge" [1].**

Keywords- Ethernet Bridge, dummynet,Ping

I. Introduction

The report outlines a simple procedure to estimate dummynet performance. We will setup a bridge as a traffic shaper to verify a variety of situations when dummynet is loaded. The goal is to be able to calibrate dummynet so that the traffic corresponds to what we want.

II. Background

This report is essentially based on [1]. The purpose of this first report was to outline a simple procedure to estimate the latency caused by a bridge. We will mainly use most of the tools and methods of this report since our work will also be to analyse the traffic going through a bridge , but with dummynet installed. We will test different scenarios to make sure that dummynet doesn't behave 'out of the ordinary'. The goal is to be able to calibrate dummynet as precise as possible so that the measured delays and packet losses introduced by it is what we want.

All the results will be helpful for further research on network simulation where dummynet might be an essential tool.

III. Description of the system

To run our tests, we will be using the same equipment setup as in [1]. We will follow the same procedure to make sure that there are no big differences with [1]. There shouldn't be any relevant difference even if the hardware is different.

A. Hardware

To run our tests, we used 3 machines: Two FreeBSD PCs and a bridge, also FreeBSD. The two first PCs were connected to each other both directly (crossover cable) or through the bridge as in figure 1.

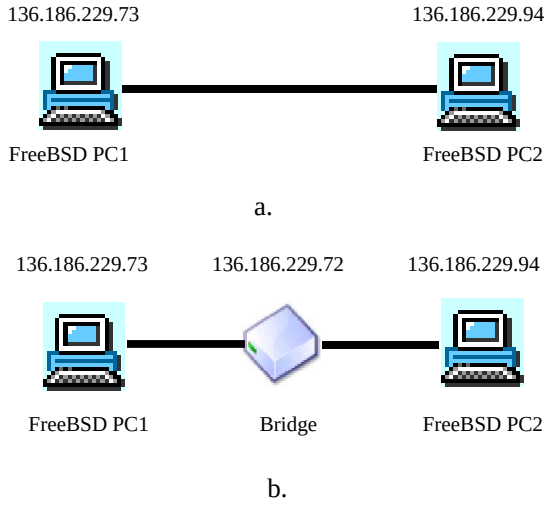

Fig 1.Latency Test Set-up

Here are the hardware details:

- The PC with IP address 136.186.229.73 sends the ping packets to the second PC. It is a mini ITX motherboard VIA C3 Samuel 2, 533.36MHz 256MB RAM running FreeBSD 4.8. It's using a VIA VT6102 Rhine II 10/100BaseTX.
- Server with IP address 136.186.229.94 replies to the ICMP ping requests from the client. It is a Compaq EVO500 P4 1.6GHz 256MB RAM running FreeBSD 4.8. It's using an Intel Pro/100 Ethernet.
- Bridge with IP address 136.186.229.72 forwards the packets, as usual. It will have dummynet installed. It is a Compaq EVO500 P4 1.6GHz 256MB RAM

running FreeBSD 4.8. It's using an Intel PRO/100 Ethernet Card and the second one is an external PCI ethernet card.

IV. Bridge Latency

As in [1]'s procedure, we first needed to characterise the behavior of the FreeBSD implementation of Ping. Then, we will run a series of tests to measure the bridge latency.

A. *Ping characterisation*

We first needed to check the performance of the ping command. This commands sends by default 64 bytes ICMP packets that are spaced 1 second apart. As an option, you can also specify different inter-packet times, that means send ICMP packets at different rates. The goal was to find out how ping command is accurate when the rate is changed.

The measurements were taken with both PC's kernel tick-time granularity set to 1000Hz. Thus to analyse ping performance we ran the following test:

- Directly connect PC1 to PC2 as in Figure1.a.
- Use tcpdump on the destination machine to capture the icmp packet. Eg: “tcpdump -i fxp0 'icmp' ”
- Send 1000 packets at different rates. Eg: “ ping -c 1000 -i 'specified wait time' dest_ip ”

The results are summarized in Fig2.

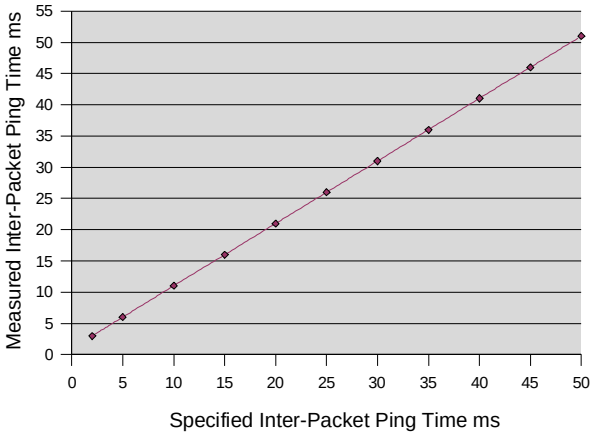

Fig 2. Inter-Packet Ping Time

As you can see, the measured ping inter-packet times are 1ms longer than what is specified in the command line and are rounded up to the next integer multiple of 1ms. This is due to the fact that the timer-tick granularity is to coarse(1ms for 1000Hz). Basically, the FreeBSD implementation of pings adds one "timer-tick" to the inter-packet time and rounds up the inter-packet time to the next integer multiple of the "timer-tick". The conclusions are the same as in report [1].

With this conclusion we should keep in mind that the ping command always adds 1ms to the specified inter-packet time (when the kernel granularity is set to 1000Hz). It won't make much difference for low rates, but for 1ms inter-packet time which in real would make 2ms, that means we changed the rate from 1000 to 500pps. For this reason, it will be impossible for further tests, using ping commands, to have accurate measurements for more than 500pps.

B. *Bridge Latency characterisation*

The bridge latency was measured as outlined in the following:

- The two PCs were connected as in Figure1b, that means they were connected to each other through the bridge. The kernel granularity was set to 1000Hz on all three machines.
- The ping packets were either 64 or 1480 bytes(actually 1480+8 bytes of ICMP header data). The measurements were conducted at different inter-packet times, from 1 to 1000ms, which corresponds to 500 to 1 pps. The number of ping packets in each test was 10000. For each rate, we did three measurements because we wanted to make sure that 'out of ordinary' RTTs would not damage the results.

The results for the ping inter-packet time at 1000ms, 10ms and 20ms are shown in table 1.

Comments	Inter-packet Time ms	Packet Size B	Min ms	Avg ms	Max ms	Std ms	Bridge Latency us
No bridge	1000	64	0.12	0.14	0.52	0.02	0
Bridge test 1	1000	64	0.19	0.22	0.63	0.01	35
Bridge test 2	1000	64	0.21	0.24	1.76	0.04	42
Bridge test 3	1000	64	0.21	0.23	0.56	0.02	41.5
No bridge	10	64	0.08	0.09	0.6	0.02	0
Bridge test 1	10	64	0.16	0.17	0.47	0.01	40
Bridge test 2	10	64	0.16	0.17	0.52	0.01	40
Bridge test 3	10	64	0.16	0.17	0.5	0.01	39.5
No bridge	20	64	0.08	0.09	0.45	0.02	0
Bridge test 1	20	64	0.16	0.18	0.5	0.02	42.5
Bridge test 2	20	64	0.16	0.18	0.49	0.02	42.5
Bridge test 3	20	64	0.16	0.18	0.61	0.02	42.5
No bridge	10	1480	0.4	0.42	0.8	0.02	0
Bridge test 1	10	1480	0.94	0.96	1.31	0.02	269.5
Bridge test 2	10	1480	0.94	0.96	1.24	0.02	270
Bridge test 3	10	1480	0.94	0.96	11.95	0.16	270.5
No bridge	20	1480	0.4	0.42	0.52	0.02	0
Bridge test 1	20	1480	0.94	0.97	1.38	0.02	269
Bridge test 2	20	1480	0.94	0.97	1.32	0.02	269
Bridge test 3	20	1480	0.94	0.97	1.41	0.02	269

Table 1. Ping RTT and Bridge Latency for 64B and 1480B packets

From Table 1, we can see for example, that at a rate of 48pps (20ms spaced packets), the average latency is 42.5us for 64Bpacket size, and at a rate of 91pps (10ms spaced packets), we have a latency of 40us . For packets of 1480B, we have an average latency of 269us for 48pps and 270us for 91pps.

To make sure that nothing 'out of the ordinary' was happening, we plotted the results of some tests. We saw consistent round trip times, with the exception of one or two outliers. Figure 3 is a plot of 64B ping packets sent at 10 and 1000ms inter-packet time.

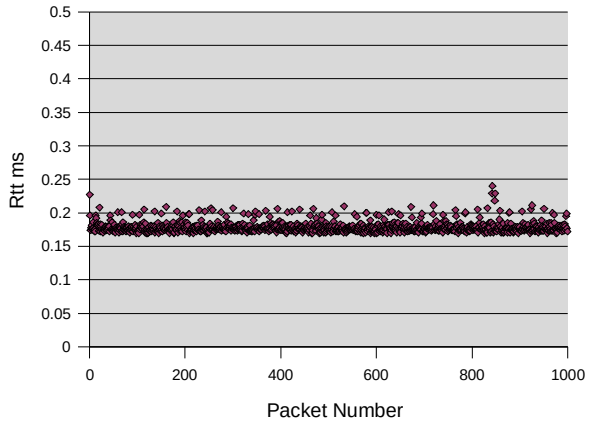

Fig 3a. Ping RTT Plots for ping 64B packets going through the bridge at 10 ms inter-packet time (91pps)

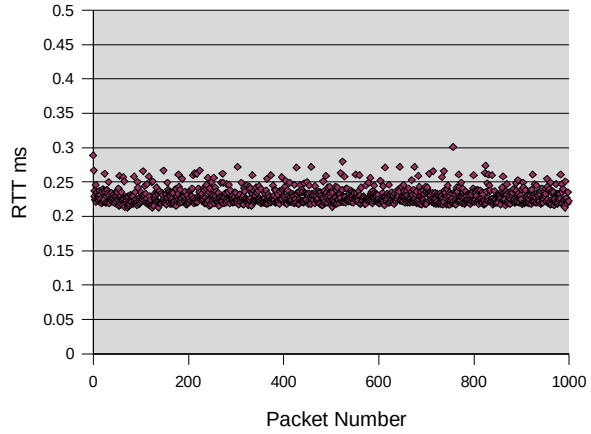

Fig 3b. Ping RTT Plots for ping 64B packets going through the bridge at 1000ms inter-packet time (1pps)

An interesting observation is that the statistics don't correspond to [1]. The bridge used in his test was working on 100 or 200MHz CPU clock. My bridge uses 1.6GHz. But apparently the latency of my bridge is higher. Even without the bridge the round trip time is higher. We thought that this additional latency might have come from the ethernet card on the client (the Rhine II 10/100BaseTX card). We thus did some further tests with another Ethernet card on the client: an Intel Pro 10/100b/100+ Ethernet. The results are in Table2.

As you can see, there is not much difference, but there is still. The latencys are a bit higher in the second case. Compared to the results in [1], without the bridge, our first setup (with the first ethernet card) has an average of 8us of additional latency and the second setup (with the Intel card) has 19us of additional latency. We haven't found the reasons for this overweight, but it will cause an overall additional latency in all the other results compared to the corresponding scenarios in [1].

We also sent ping packets with other rates to see how it affects the latency. Figure 4 shows how the bridge latency changes compared to the inter-packet time. As you can see, the rate doesn't influence the latency. The latency doesn't change that much from one rate to another.

Comments	Wait Time ms	Packet Size B	Min ms	Avg ms	Max ms	Std ms	Bridge Latency us
No bridge	1000	64	0.13	0.15	1.04	0.02	0
Bridge test 1	1000	64	0.21	0.23	0.92	0.01	41.5
Bridge test 2	1000	64	0.21	0.23	0.38	0.01	41.5
Bridge test 3	1000	64	0.21	0.23	1.1	0.02	41.5
No bridge	10	64	0.09	0.1	0.61	0.01	0
Bridge test 1	10	64	0.17	0.18	0.39	0.01	39
Bridge test 2	10	64	0.17	0.18	0.29	0.01	39
Bridge test 3	10	64	0.17	0.18	0.96	0.02	39
No bridge	20	64	0.09	0.11	0.54	0.01	0
Bridge test 1	20	64	0.17	0.18	0.25	0.01	38.5
Bridge test 2	20	64	0.17	0.18	0.59	0.01	38.5
Bridge test 3	20	64	0.17	0.18	0.69	0.01	38.5
No bridge	1000	1480	0.43	0.45	0.6	0.01	0
Bridge test 1	1000	1480	0.98	1.01	1.59	0.02	280.5
Bridge test 2	1000	1480	0.95	1.01	2.83	0.02	280.5
Bridge test 3	1000	1480	0.98	1.01	1.7	0.02	280.5
No bridge	10	1480	0.38	0.4	0.91	0.01	0
Bridge test 1	10	1480	0.93	0.94	1.51	0.01	271.5
Bridge test 2	10	1480	0.93	0.94	11.94	0.11	271.5
Bridge test 3	10	1480	0.93	0.94	1.8	0.01	271.5
No bridge	20	1480	0.38	0.4	1.13	0.02	0
Bridge test 1	20	1480	0.93	0.94	21.92	0.21	272
Bridge test 2	20	1480	0.93	0.95	1.71	0.02	271
Bridge test 3	20	1480	0.93	0.95	21.95	0.21	272

Table 2. Ping RTT and Bridge Latency for 64B and 1480B packets with the second ethernet card

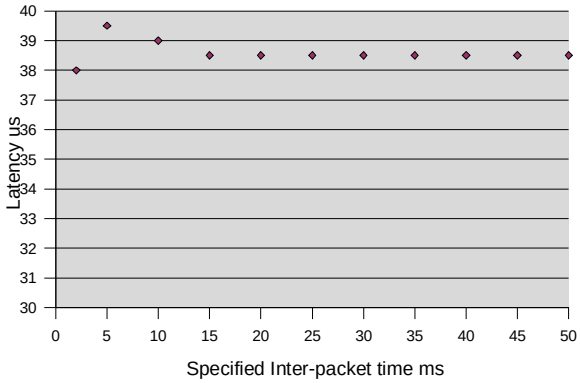

Fig 4: Bridge Latency versus inter-packet time for 64B packets.

We also did the same tests at higher rate to make sure that nothing would go 'out of the ordinary'. Again the number of packets sent is 10000. The results for 1 and 2ms inter-packet time, resulting respectively to 500 and 333pps are shown in table 3.

As you can see, when we compare with Table 2, the average latency for higher rates is the same. For 64 bytes packets, the latency stays around 42us and for 1480 bytes packets it is around 270us, which was the same for lower rates (inter-packet times between 1000 and 5ms). Thus we can conclude that modest packet rates do not affect bridge latency.

Comments	Wait Time ms	Packet Size B	Min ms	Avg ms	Max ms	Std ms	Bridge Latency us
No bridge	1	64	0.09	0.09	0.22	0.01	0.0
Bridge test 1	1	64	0.17	0.18	6.52	0.1	41.5
Bridge test 2	1	64	0.17	0.18	0.3	0.02	45
Bridge test 3	1	64	0.17	0.18	0.72	0.02	43.5
No bridge	2	64	0.09	0.1	0.51	0.01	0.0
Bridge test 1	2	64	0.17	0.18	0.61	0.02	41
Bridge test 2	2	64	0.17	0.18	0.53	0.02	41.5
Bridge test 3	2	64	0.17	0.18	0.66	0.02	41.5
No bridge	1	1480	0.38	0.4	5.63	0.13	0.0
Bridge test 1	1	1480	0.92	0.94	2.79	0.04	271
Bridge test 2	1	1480	0.92	0.94	1.09	0.01	269
Bridge test 3	1	1480	0.92	0.94	1.28	0.02	269.5
No bridge	2	1480	0.38	0.4	4.18	0.09	0.0
Bridge test 1	2	1480	0.92	0.94	3.94	0.04	269.5
Bridge test 2	2	1480	0.92	0.94	3.92	0.04	270
Bridge test 3	2	1480	0.92	0.94	3.92	0.04	271

Table 3. Ping RTT and Bridge Latency for 64B and 1480B packets at high rates

Figure 5 and 6 show the cumulative relative frequency for 64B and 1480B pings packets at different rates (2, 10, 20, 1000ms equivalent to 333, 91, 49, 1pps), without and with the bridge in the middle. These tests were run with the second Ethernet card on the client (the Intel Pro 10/100b/100+ Ethernet card) .

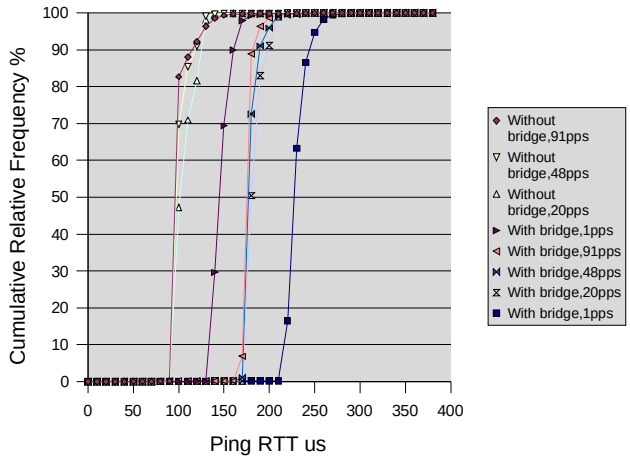

Fig 5:Round Trip Latency Cumulative distribution for 64B packets

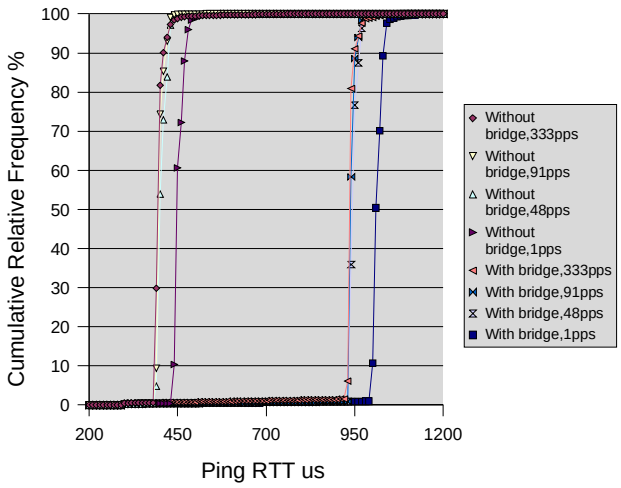

Fig 6:Round Trip Latency Cumulative distribution for 1480B packets

V. DUMMYNET

FreeBSD's dummynet uses the kernel-resident ipfw (firewall) functionality to select packets that are to be subjected to packet delay, random loss or bandwidth limitation. For our tests, we forced all packets going from the client to the server and vice versa to go through a 'pipe'. Those pipes will define what delay, loss and bandwidth the packets are subjected to.

A. Dummynet Latency

First we wanted to know if the latency was increased when dummynet was installed on the bridge. The bridge latency was again measured, as follows:

- Both client and server's kernel granularity are changed to 1000HZ.
- Dummynet was loaded. Pipes were created for both ways (client to server and server to client) and configured so that the packets going to the bridge suffered no delay. Eg: “ipfw add pipe 1 ip from any to any bridged” and “ipfw pipe 1 config delay 0ms”
- 64B ping packets were sent at different rates: The inter-packet time varied from 1 to 1000ms. The number of packets in each test was 10000. We ran the test twice for each combination.

The results are gathered in Table 4.

Comments	Wait Time ms	Packet Size B	Min ms	Avg ms	Max ms	Std Dev ms	Bridge Latency us
No bridge	1000	64	0.13	0.15	1.04	0.02	0
Dummynet test 1	1000	64	0.21	0.24	0.81	0.01	53
Dummynet test 2	1000	64	0.21	0.24	0.6	0.01	53
No bridge	10	64	0.09	0.1	0.61	0.01	0
Dummynet test 1	10	64	0.17	0.18	0.82	0.01	44.5
Dummynet test 2	10	64	0.17	0.18	0.35	0.01	44.5
No bridge	2	64	0.09	0.1	0.51	0.01	0
Dummynet test 1	2	64	0.17	0.18	0.33	0.01	43.5
Dummynet test 2	2	64	0.17	0.18	0.53	0.01	44
No bridge	1	64	0.09	0.09	0.22	0.01	0
Dummynet test 1	1	64	0.17	0.18	0.72	0.02	44
Dummynet test 2	1	64	0.17	0.17	0.75	0.01	43.5

Table 4: Ping RTT and Bridge Latency with ipfw and Dummynet installed

As you can see, the rate doesn't influence the delay that much: compared to the average bridge latency without dummynet installed, the difference is of the order of 2-8us.

Installing dummynet and ipfw on the bridge increased the latency from 41.5us to 53us for 1pps (1000ms), from 39us to 44.5us for 91pps (10ms), from 41us to 44us for 333pps (2ms) and from 42us to 44us for 500pps (1ms) for 64B packets.

Figure 7 shows the cumulative frequency distributions for 10,20,1000ms inter-packet time (91, 48 and 1pps) without and with.

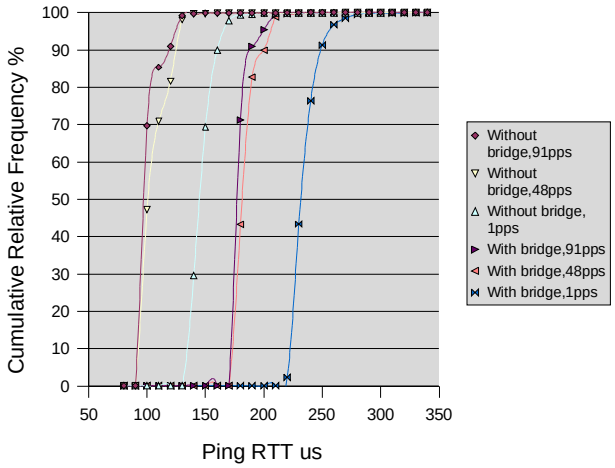

Fig 7:Round Trip Delay Cumulative Distribution for 64B packets at 91,48,1pps

B. *Dummynet Delay*

The first option that we can control with dummynet is the delay on the traffic. This is mainly used to simulate different distance between machines. We thus need to check the accuracy of dummynet specified delays. The bridge latency was again measured, as follows:

- Dummynet delay was varied from 1000ms to 1ms at 1pps packet rate. Eg: " ipfw pipe 1 config delay 'delay'".
- Ping packets were sent from PC1 to PC2. The number of packets in each test was 10000.

The results for 1pps rate are summarized in Table 5.

Dummynet Delay ms	Min ms	Avg ms	Max ms	Std Dev ms	One-Way Delay us
1	1.1	1.61	2.16	0.29	740
2	3.1	3.61	4.17	0.29	1740
5	9.1	9.61	11.03	0.29	4740,5
10	19.1	19.61	20.15	0.29	9739
25	49.1	49.6	50.15	0.29	24737.5
50	99.09	99.6	100.14	0.29	49734.5
75	149.09	149.59	150.14	0.29	74731.5
100	199.08	199.59	201.04	0.29	99728.5
500	998.98	999.49	1007.74	0.3	499682
1000	1998.86	1999.37	1999.89	0.29	999619

Table 5: Dummynet Delays Statistics at a rate of 1pps

As you can see the higher is the delay, the more precise is dummynet. For 1ms of delay, the measured delay is 0.74ms which is 74% of what is specified. For 50ms, the measured delay is 49.734ms which is 99% precise.

C. *Dummynet Delay versus rate*

This first tests were conducted at a regular rate of 1pps. To make sure that dummynet supports higher rates, we ran the same test with different rates. If dummynet is stable, the real delay should not vary with the rates. For each dummynet delay chosen like in Table 5, we varied the inter-packet time from 1ms to 500ms. The number of ping packets sent for each test was 10000. The results are in table 6.

Inter-pack ms / Delay ms	500	333	200	66	40	20	10	2
0	0.0	0.0	0.0	0.0	0.0	0.1	0.1	0.1
1	0.8	0.7	0.8	0.7	0.8	0.8	0.8	0.8
2	1.8	1.8	1.8	1.8	1.7	1.7	1.8	1.8
5	4.7	4.7	4.8	4.7	4.8	4.8	4.8	4.8
10	9.7	9.8	9.7	9.8	9.7	9.7	9.8	9.8
25	24.7	24.8	24.7	24.8	24.7	24.8	24.8	24.8
50	49.7	49.8	49.7	49.8	49.7	49.7	49.8	49.8
75	74.8	74.7	74.8	74.7	74.7	74.7	74.8	74.8
100	99.8	99.8	99.8	99.7	99.7	99.8	99.8	99.8
500	499.6	499.7	499.7	499.7	499.7	499.7	499.7	499.7
1000	999.6	999.6	999.6	999.6	999.6	999.6	999.7	999.7

Table 6: Real delay in us versus Dummynet specified delay (ms) and inter-packet time ms

As you can see, the real delay is not influenced by the rate. It stays pretty much stable. At any rate, the measured delay is what we expected. Figure 8 is a graphical result of the measured delay versus the specified delay and the rate, for delays from 1 to 50 ms and packet rates from 2 to 500pps. Clearly, dummynet doesn't suffer from different packet rates traffic, the measured delays are almost the same (1us of precision): all the lines (each corresponding to one rate) are one over each other.

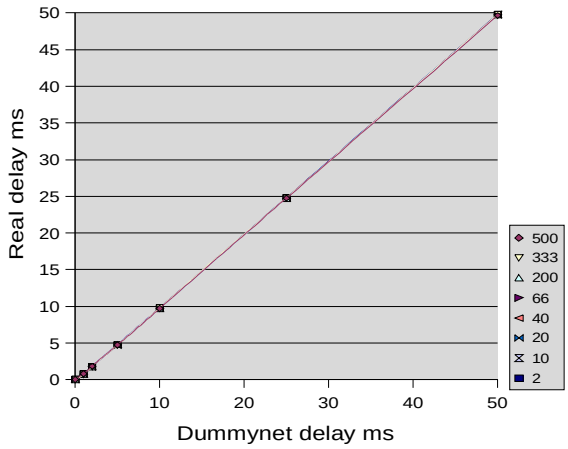

Fig 8: Real delay versus specified delay and rate

D. *Dummynet Packet Loss*

Another option that we can control with dummynet is the packet loss on the traffic. The packet loss was measured, as follows:

- Dummynet random loss was varied from 2 to 95% at 1pps packet rate. Eg: " ipfw pipe 1 config plr 'packet_loss'.

- Ping packets were sent from PC1 to PC2. The number of packets in each test was 10000.

We did 3 runs for each probability at 1pps, plus some at different rates (50,25,15,5,2,1ms inter-packet time). The results are in table 7.

Dummynet Loss(%)	Real loss rate (%)	packet interval (ms)	Dummynet Loss(%)	Real loss rate (%)	packet interval (ms)
2	1.7	1000	35	34.7	15
2	1.5	1000	35	32.3	5
2	2.93	1000	35	35.43	2
2	1.5	50	35	35.31	1
2	2.2	15	50	50.4	1000
2	0.9	25	50	52.2	1000
2	1.8	5	50	46.4	1000
2	2.05	2	50	48.8	50
2	2.09	1	50	50.1	25
5	4.3	1000	50	50.2	15
5	4	1000	50	48.1	5
5	3.9	1000	50	50.08	2
5	4.8	50	50	50.55	1
5	4.7	25	65	66.53	1000
5	5.2	15	65	64	1000
5	5	5	65	65.7	1000
5	5.31	2	65	64.96	50
5	5.14	1	65	63.6	25
7	6.5	1000	66	65.8	15
7	7.4	1000	66	64.53	5
7	7.6	1000	65	64.89	2
7	7.62	50	65	65.64	1
7	6.9	25	85	84.38	1000
7	8.8	15	85	85.6	1000
7	7.71	5	85	85.54	1000
7	5.14	2	85	84.6	50
7	5.36	1	85	85.03	25
15	14.7	1000	85	84.66	15
15	16.52	1000	85	84.05	5
15	14.4	1000	85	84.6	2
15	15.7	50	85	84.47	1
15	15.1	25	95	95.97	1000
15	14.4	15	95	95.15	1000
15	13.8	5	95	95.15	1000
15	15.17	2	95	95.19	50
15	14.87	1	95	94.55	25
35	35.87	1000	95	94.76	15
35	33.5	1000	95	95.08	5
35	34.2	1000	95	94.74	2
35	35	50	95	95.04	1
35	34.8	25			

Table 7: Dummynet Packet Loss Statistics at a rate of 1pps

From Table 7, we can conclude that dummynet probability of loss is pretty accurate. Dummynet achieves in average 98% precision on packet loss.

We also wanted to check how the packet loss is distributed over time. This will show how random dummynet forces the packet to get lost. In Figure 9 we ran dummynet with a packet loss of 7% at different packet rates to see how random the packet are lost. Each row is one run at a specific rate (1pps three times, 17 and 38pps). Each data point is a 'loss event'. What we want to see is if there is any persistent cluster at a specific time, or if the packets are just lost randomly at no particular moment (or packet number), this might be the case when the loss depends also on the packet queue waiting to be sent through the pipe. Our conclusion is that the distribution looks pretty random: there is no burst in the loss distribution.

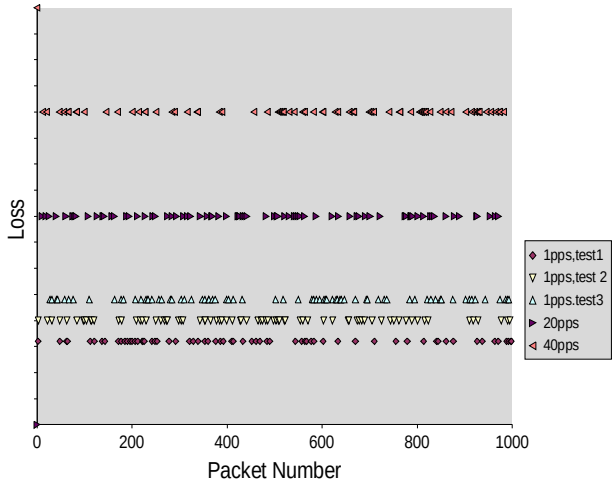

Fig 9: Packet Loss distribution for 7%loss at 1,20 and 40pps

VI. Conclusion

We can build an Ethernet Bridge from a simple PC with minimum 2 Ethernet cards. By loading dummynet on this bridge, we can easily implement a simple network simulator .

The RTT of 64 and 1480 bytes ping packets was used to obtain an estimate of the average bridge latency. Using this method we found that the average bridge latency was 39us for 64B packets and 275us for larger packets (1480B).

As a traffic shaper, we installed dummynet on the bridge which is a simple tool to select packets that we want to delay or who should suffer packet loss. The overhead introduced by loading dummynet increased the average latency from 39 to 48us for 64B packets.

Dummynet is pretty accurate concerning delays and packet loss, the results match our requests: the measured delays and packet loss corresponds to the delays and packet loss configured in dummynet's setup. Dummynet is thus a satisfying and simple tool for network simulations.

Acknowledgements

Lawrence Stewart and Claudio Favi provided a big support on the installation of the testbed, their help was much appreciated. I am also grateful for the assistance that G. Armitage gave me in preparing this report.

References

[1] A.L. Cricenti, "Evaluation of a Pentium PC for use as an Ethernet Bridge", CAIA Technical Report 030326A, March 2003, http://caia.swin.edu.au/reports/030326A/CAIA-TR-030326A.pdf.

[2] L.Rizzo "Dummynet" http://info.iet.unipi.it/~luigi/ip_dummynet.

[3] L.Rizzo, "Dummynet: a simple approach to the evaluation of network protocols", http://info.iet.unipi.it/~luigi/dummynet.ps.gz.

[4] "FreeBSD handbook, http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/

Appendix 1 – Network setup

Outlined below are the configuration changes required to setup the network. That is: the 2 machines and the bridge with dummynet and ipfw.

A. Client and server software configuration

1. In order to test our bridge at high rates you need to make sure that the destination, PC2 in our case, doesn't limit the number of icmp incoming packets. This limit is usually used to protect your machine from unwanted packet floods. Do the following:

   %sysctl net

   to check the kernel variable concerning the network.. And to change the variable, do, as root:

   %sysctl net.inet.icmp.icmplim=10000

2. To setup authentication between machines, you need to create key pairs. This will allow you to ssh as root to the remote machine without having to give any password. First, edit /etc/sshd_config on the machine you want to ssh too, the remote machine (PC2). Set "PermitRootLogin variable to YES, and uncomment the line. Than do the following on your machine(PC1):

   % ssh-keygen -t rsa on the machine you want to ssh from.

   %scp /root/.ssh/id_rsa.pub root@PC2:/root/.ssh/.

   On PC2, do:

   %cd /root/.ssh

   %cat id_rsa.pub >> authorized_keys

   You need to do this to each pair of machines you want to ssh as root from one to the other.

B. Bridge software configuration

1. To automatically load the bridge kernel module at boot-up edit the /boot/loader.conf file to include the following:

   bridge_load="YES"

   For dummynet, you need to add the ipfw module, thus add to the same file:

   ipfw_load="YES"

2. Loading the dummynet module using the above mechanism doesn't work. However, we can run a shell script at boot-up which will load the dummynet kernel module. You thus have to create a shell script in /usr/local/etc/rc.d/ with the following line

   kldload dummynet

   With this configuration, you should still not be able to use your bridge with dummynet. This is because ipfw has a default law that deny any packet from any to any. In the same script add these line to create the pipes you will use. Make sure the shell script is executable. That is for example:

   ipfw add pipe 1 ip from any to any bridged

   ipfw add allow ip from any to any

3. Dummynet performs its task once per timer tick; since the default granularity is 100Hz, this limits delays to be larger than 10ms. You need to rebuild the kernel. The kernel configuration is set in /usr/src/sys/i386/conf/. Create a new kernel configuration file (copy the default one), and add this line to it and then run the the following:

   options HZ=1000 #sets timer granularity to 1000Hz.

   %config NEWGENERIC

   %cd ../../compile/NEWGENERIC

   %make depend && make && make install

4. Add the following to /etc/sysctl.conf

   #for bridging

   net.link.ether.bridge=1

   net.link.ether.bridge_cfg=if1:x.if2:x

   where if1:x refers to the interface and cluster on which bridging is enabled, in our case: net.link.ether.bridge_cfg=fxp0:1,rl0:1

   #for ip firewall

   net.link.ether.bridge_ipfw=1

   #for dummynet

   net.link.ip.fw.enable=1

   net.inet.ip.fw.one_pass=1

5. Finally add these lines to /etc/rc.conf

   gateway_enable="YES"

   firewall_enable="YES"

   firewall_script="/etc/rc.ipfw"

6. Change /etc/rc.ipfw to configure your pipes. Add:

   ipfw pipe 1 config delay 0ms

   This can also be issued from the command line.

You should know be able to use the bridge as a traffic shaper. To change dummynet configuration, such as the delay or packet loss, do:

ipfw pipe 1 config delay 10ms

ipfw pipe 1 config plr 0.15

Appendix 2 – Procedure

a) Get the files

This is the procedure used to run all those tests by yourself. A bunch of scripts have been written to fasten the process. You need to get NETTESTS directory from mordor:

%mkdir NETSIM

%scp -r mordor:/home/wendyv/NETTESTS NETSIM/.

Note: mordor's ip address: 136.186.229.17

You should have different scripts and four folders: data, logs, src and bridgeSetup. Data will gather all the measurements results and logs all the ping logs mainly. Src contains all the source c programs and bridgeSetup has all the files that you will need to copy into your bridge. The first group of script you will use are the ping scripts which run a series of pings with different situations. These are: nobridgeWait.sh, noBridgeLatency.sh, LatencyAll.sh.

The second group are the scripts who will read the logs and gather the results into the data folder. These are: WaitResults.sh, LatencyResults.sh, DelayResults.sh, LossResults.sh, CumFreqResults.sh. These files use c programs located in the src folder: readWait.c, readLatencyAll.c, readDummynetDelay.c readDummynetLoss.c

b) Get the measurements and results

The first tests should be run without the bridge in the middle, that means PC1 should be connected directly to PC2 (see Figure 1a). you need to define a client and a server. The client should have all the source code copied from mordor.

1. First connect the client to the server with a crossover cable. You need to make sure that they don't have a default route to another machine, if it does do: “route delete default” as root one both machines. You can check with “netstat -rn” if everything is ok.
2. Check that you can ping from one to the other. Try to ssh also.

   The following makefile will run all the pings necessary to have sufficient data for our measurements. It will then gather the results into one file. You first need to edit the makefile and change some variable so that it corresponds to your settings (such as ip addresses, directory name, etc..). Then run as root: % make -f Makefile_noBridge
3. There should be a new file in the 'data' folder: WaitTime.txt which gives a table corresponding to the ping specified versus the measured inter-packet time (needed for ping characterisation as in chapter IV.A.

Now we will do the rest of the tests with the bridge installed between the two other machines.

1. Connect the client to the bridge and bridge to server with a crossover link. Since you might only have one ip address for the bridge (the 3 computers are on the same subnetwork), it is better if you put the ipaddress visible to the client.
2. Setup the bridge as explained in [1] Appendix 2. But don't install ipfw and dummynet yet. Just setup the bridge.
3. Create a folder on the bridge which should have the same path as in your client. Copy all the files that are in bridgeSetup into this folder.
4. Make sure you can ping and ssh between the client to the two other machine, especially to the bridge, and from the server to the client. For all these pairs, you need to have root login access and ssh authorization to login without a password (you need to generate keys, see Appendix 1).
5. Edit and change the following makefile, in order to correspond to your settings. This makefile runs the pings, writes some logs, gathers the results and creates simple tables with all the data. Run as root: % make -f Makefile_withBridge
6. You should have new files in the data folder: Latency.txt , Cumulator.txt, dataFlood.txt, DummynetDelay.txt, LossPlot.dat, DummynetLoss.txt, WaitTime.txt plot.dat

Latency.txt compares the ping ran with the bridge to the ones without bridge. DummynetDelay.txt is more concerned with dummynet delays depending on rate, and specified delay. DummynetLoss.txt looks at the packet loss accuracy of dummynet, comparing specified packet loss to real packet loss percent. Cumulator text files are the data for the Cumulative Relative Frequency plots (Figure 5, 6, 7). LossPlot.dat and plot.dat are for plotting purposes (Figure 8, 9). All the logs of the ping requests are in the 'logs' folder. You should have 3 folders in there : NOBRIDGE for the pings tests without a bridge in the middle, BRIDGE1000HZ for the tests with the bridge installed but not dummynet, and DUMMYNET, for the dummynet measurements.