Evaluation of a Pentium PC for use as an Ethernet Bridge

A.L. Cricenti
Centre for Advanced Internet Architectures. Technical Report 030326A
Swinburne University of Technology
Melbourne, Australia
tcricenti@swin.edu.au

***Abstract*-This technical report describes the construction of an inexpensive Ethernet Bridge built from an obsolete Pentium PC (IBM PC 300GL Pentium 166MMX) and Ethernet cards. The report outlines a simple procedure for estimating the latency of the bridge. Results are presented for the time-stamping accuracy of the bridge, the average latency and FreeBSD ping characteristics. The bridge can be used as a traffic shaper by installing FreeBSD's dummynet module.**

Keywords- Ethernet Bridge, Dummynet, Ping, Latency, CAIA

I. Introduction

This report outlines the experience gained during the construction and configuration of an inexpensive Pentium PC based Ethernet bridge and outlines a simple procedure to estimate the latency of the bridge. This bridge can be used as a traffic monitor and or traffic shaper. The bridge is flexible enough so that it can be used in a variety of situations for example: to capture packet traces from home Internet users, or to act as a traffic shaper for studying the effects of packet loss and delay.

II. Description of the Ethernet Bridge

The Ethernet Bridge is built from a Pentium PC running FreeBSD 4.6 Release 0. The purpose of this bridge is twofold; the first is to collect packet traces from home Internet users. The bridge is designed to be connected between either an ADSL or cable modem, (or other Ethernet device) and the user's PC or small home LAN. The second purpose of the bridge is to act as a variable latency device, which can be used to study the effect of latency and packet loss on the performance of network game players. In this case, the bridge is placed between two game consoles (see Figure 1) and it is used to selectively delay or drop packets. FreeBSD is a good choice for the operating system of the bridge as it has both bridging support[1] and ipfw/dummynet[2], the latter being useful for controlled packet delay/loss.

A. Bridge Hardware

Having obtained a used IBM PC 300GL Pentium 166MMX relatively cheaply, we decided to use this machine as the basis of an Ethernet Bridge. The PC had 64MB RAM and a 1.7GB HDD. We installed FreeBSD 4.6. The overall the cost of the hardware was less than $100. The final configuration of the bridge has 3 Ethernet network interface cards, two of which, tx0 and

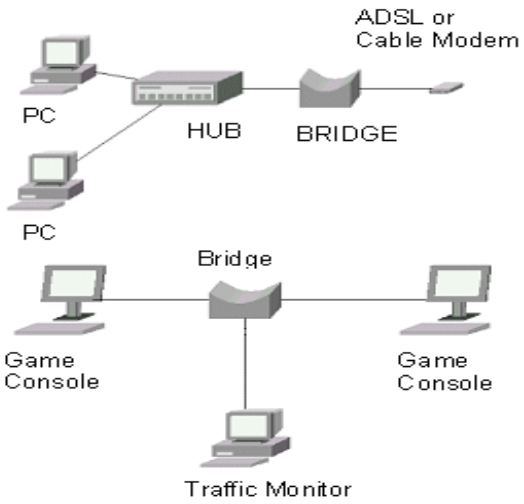

Figure 1 Bridge Deployment

tx1, are fast Ethernet cards (SMC EtherPower II 10/100). These fast Ethernet adapter cards are used for bridging. The third adapter (cs1) is a 10BaseT adapter (CS8920 Ethernet Adapter) that is built into the PC's motherboard; we chose to use this interface to remotely access or configure the bridge via IP. This card was not used for bridging as it does not support 100Mbps bit rates. Installing FreeBSD 4.6 requires at least 16MB of RAM; however, it can be run with 4-8MB of RAM after installation [3]. The minimum hard drive size required is 100MB [4], but realistically the hard drive should be at least 250 to 350 MB. If the bridge is to be used to capture and store packet traces then a larger hard disk is desirable[1].

Appendix 1 contains a more detailed description of the bridge hardware.

B. FreeBSD Configuration

FreeBSD was chosen as the operating system for the following reasons:

- It has all the necessary code for bridging support.
- It has a robust IP stack.

1 In our target application, compressed packet traces from a home Internet user are approximately 3 to 5 MB per day, therefore a 1GB drive would be suitable to store in excess of one month of packet traces.

- It has tools for bandwidth management and controlled packet delay/loss (dummynet and ipfw) Dummynet simulates/enforces queue and bandwidth limitations, delays, packet losses, and multipath effects. It also implements a variant of Weighted Fair Queuing called WF2Q+.
- It supports standard UNIX tools for collecting packet traces (tcpdump).

To enable kernel support for bridging and dummynet, we chose to load bridge, ip firewall (ipfw) and dummynet as kernel modules by issuing the "kldload bridge", "kldload ipfw" and "kldload dummynet" commands. (Alternatively, the kernel can be recompiled with the appropriate options.) Bridging, Dummynet and ipfw also require changes to the rc.conf and sysctl.conf files to enable bridging, ip firewall and to point to the firewall rule script. Appendix 2 contains the details of the required changes.

III. Bridge Latency

A simple method to measure the bridge latency was developed using ICMP Ping packets and standard PCs. The procedure is outlined in the following section.

A. *Equipment Set-up*

Two PCs, configured as below, were connected to each other both directly (crossover cable) and through the bridge (see Figure 2).

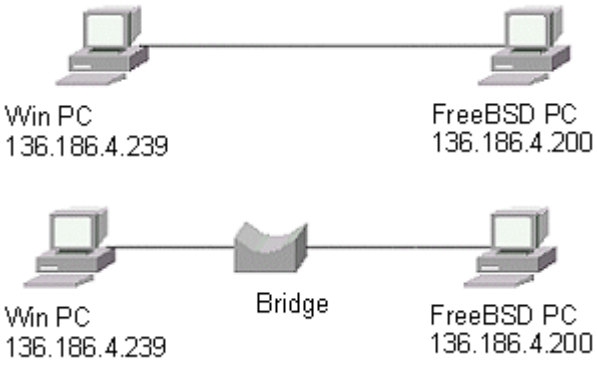

Figure 2 Latency Test Set-up

- The PC with IP address 136.186.4.200 (FreeBSD) was a Compaq EVO5000 P4 1.7GHz 256MB RAM running Free BSD 4.6. This PC was used as the source of the ping packets.
- The PC with IP address 136.186.4.239 (WinPC) was a Compaq EVO5000 P4 1.7GHz 256MB RAM running Windows 2000.
- The clock speed of the bridge's CPU could be varied from 100MHz to 200MHz by setting the appropriate dip switches on the motherboard. This feature was useful as it allowed the bridge latency to be measured at different CPU speeds.

B. *Time-stamping Accuracy*

The bridge is to be used to collect packet traces, therefore we need to assess the time-stamping accuracy of the bridge. The following procedure, was used to measure the time-stamping accuracy of the bridge.

- Use Netcom System Smartbits 2000 to generate ethernet frames with the following characteristics:
- Frame: 92 bytes Ether/IP/UDP + 4 byte CRC
- Fixed-space intervals: 125µs
- Link Speed: 100Mbps
- Frames were captured at the bridge using:

  "tcpdump -n -i tx0 -w <file>"
- Trace files were read with:

  "tcpdump -n -ttt -r <file>"

The Smartbits inter-frame time is measured from the end of the frame to the beginning of the next frame. On the other hand, the difference between consecutive tcpdump timestamps is measured from the beginning of a frame to the beginning of the next frame. The two times, therefore, differ by the frame duration. Since the frames are 104 bytes (96 bytes plus 8 preamble) long, then the duration of the frame is $104 \times 8/100 \approx 8.3$µs. Thus the time from the beginning of one frame to the next is 125+8.3 = 133.3µs.

The results obtained from the bridge are summarised in Table 1.

CPU Clock MHz	*Mean*	*Median*	*Std Dev*	*No of frames*	*Max*	*Min*
100	133.3	133	6.17	65535	339	37
200	133.3	133	3.38	65535	305	41

Table 1 Tcpdump Time-stamping Accuracy

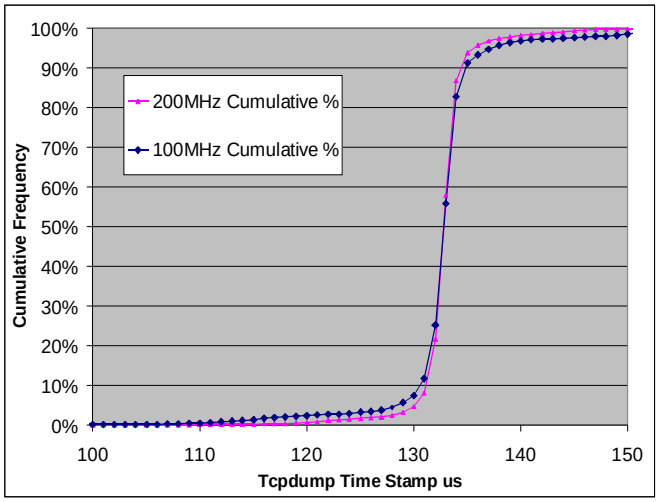

Figure 3 Ping RTT Cumulative Distribution

We see that increasing the CPU clock speed reduces the variability in the time-stamps. The cumulative distribution, Figure 3 shows that 95% of the time-stamps lie in the range 131 to 135µs for the 200 MHz CPU clock case and 128 to 139µs for the 100 MHz CPU clock case. Approximately 98% of the time-stamps are within ±10% of 133µs (200 MHz); whilst at the slower CPU clock, 95% of the time-stamps within ±10% of 133

µs. This error corresponds to ±27 bit times at 2 Mbps (cable modem rates) so we can say that the time-stamping accuracy of the bridge is adequate for measuring packet inter-arrival times for home Internet traffic;.

C. Ping Charaterisation

As ping packets were used as the basis for estimating the Bridge latency, we needed to characterise the behaviour of the FreeBSD implementation of Ping.

The FreeBSD 4.6 implementation of Ping defaults to sending 64 byte ICMP packets that are spaced 1 second apart. The "– i N" option can be used to set the time between ping requests to "N" seconds. This inter-packet interval can be less than one second if the ping is issued by the super-user (root).

To characterise the FreeBSD ping performance we ran the following test to measure the time between ping packets:

- Directly connect the FreeBSD PC to the WinPC.
- Use tcpdump on the FreeBSD machine to capture the ICMP packets. Eg: tcpdump -i fxp0 -w <file> 'icmp'
- Issue the ping –c 500 –i "specified wait time" command
- Take the average of the 500 time intervals.
- The tests were run only once at each inter-packet time.

We wanted to check the implications of changing the kernel timer-tick granularity on the ping inter-packet times. The default kernel granularity FreeBSD 4.6 is 10ms (100HZ); the granularity can be made finer by recompiling the kernel with the "HZ option" set to a higher value. Increasing the kernel "HZ option" to 1000Hz reduces the timer-tick time to 1ms.

The results obtained for the cases where the kernel granularity was set to 100HZ and 1000Hz are summarised in Figure 4.

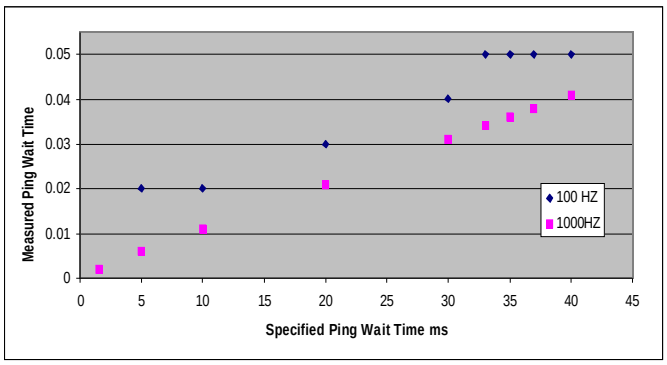

Figure 4 Ping "Wait" Time

We see from Figure 4 (100Hz) that the measured ping wait times are 10ms longer than what is specified in the command line (wait +10ms). The wait times are also rounded-up to the next integer multiple of 10ms. The rounding does not affect the RTT time-stamping. So for a specified wait time of 5ms, the actual wait time becomes 20ms. This is due to the fact that the timer-tick granularity is too coarse (10ms or 100Hz). With the kernel granularity set to 1000Hz, the measured ping wait times are 1ms longer than what is specified in the command line (wait +1ms). The wait times are also rounded-up to the next integer multiple of 1ms. The FreeBSD implementation of ping adds one "timer-tick" time to the wait time and rounds up the wait time to the next integer multiple of the "timer-tick".

D. Flood Ping

The flood ping (ping –f) outputs ping packets as fast as they can be generated with a minimum of 100 packets per second. To determine the packet rate of the flood ping, we used the following procedure to measure the inter-packet time.

- Directly connect the FreeBSD PC to the WinPC.
- Use tcpdump on the FreeBSD machine to capture the ICMP packets. Eg: tcpdump -i fxp0 -w <file> 'icmp'
- Issue the ping –c 56636 –f command from the FreeBSD PC.
- The mean and median times between ping requests are shown in Table 2. Using the median time between ping requests, the flood ping sends packets at a rate of 9.4 kpps (≈4Mbps) for 64 byte pings and 2.8 kpps (≈30Mbps) for the larger 1480 byte pings.

Ping Packet Size	*Mean ms*	*Median ms*	*Std Dev*	*Rate pps*
64 Bytes	109.97	106.00	32.69	9434
1480 Bytes	361.31	358.00	20.82	2793

Table 2 Flood Ping Statistics

E. Bridge Latency Measurements

The bridge latency was measured as is outlined in the following:

Ping packets were sent from the FreeBSD PC (136.186.4.200) to the Win PC (136.186.4.239) both with the bridge in place and with the bridge removed see Figure 2.

These measurements were taken only with the bridge kernel module loaded (i.e. before dummynet and ipfw were installed). The kernel granularity of the bridge was set to 1000Hz[2].

The ping packets were either 64 or 1480 bytes spaced at 20ms (ie ping –c 100000 –i 0.01 136.186.4.200), resulting in a packet rate of 50 pps (approx 50 kbps and 500 kbps). The 20ms interval was chosen since the kernel granularity of the ping source (ie the FreeBSD PC) was 10ms. The number of ping packets in each test was 100000. The average round trip time (RTT) was used to estimate the bridge latency using the following formula:

Bridge Latency = (Average RTT with bridge – Average RTT without bridge)/2

The ping results for the tests were plotted (Figure 5 shows some representative plots) to ensure that nothing "out of the ordinary" was happening. The graphs show that most of the ping packets are delayed by roughly the

2 The default kernel granularity of 100Hz gives similar results.

same amount with the exception of one or two outliers. This behaviour is deemed acceptable as it occurs both with the bridge and without it. No ping packets were lost during these tests.

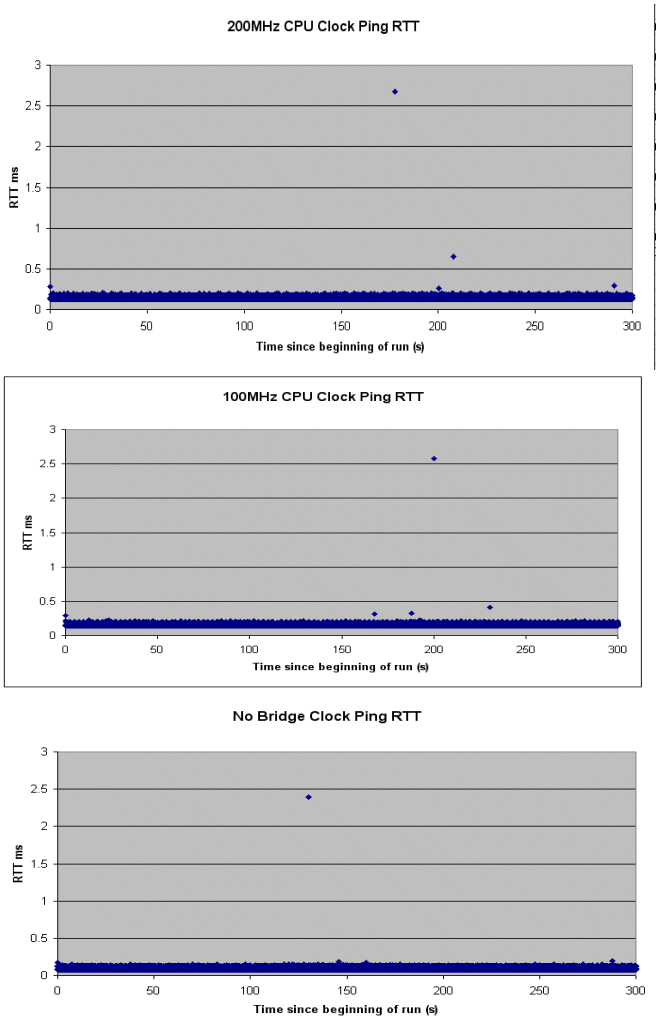

Figure 5 Ping RTT Plots

The round-trip times in ms and average bridge latency in µs are shown in Table 3.

Comments	*CPU Clock MHz*	*Min ms*	*Average ms*	*Max ms*	*Std Dev*	*Bridge Latency µs*
No Bridge	NA	0.077	0.082	2.660	0.015	0
Bridge Test 1	100	0.137	0.146	2.577	0.013	32
Bridge Test 2	100	0.136	0.145	2.227	0.012	31.5
Bridge Test 3	100	0.136	0.144	1.957	0.013	31
Bridge Test 1	200	0.124	0.131	2.674	0.013	24.5
Bridge Test 2	200	0.124	0.131	2.092	0.014	24.5
Bridge Test 3	200	0.124	0.131	1.390	0.010	24.5

Table 3 Ping RTT and Bridge Latency

From Table 3, the average latency is 24.5µs at 200MHz and rises to 32µs with a 100MHz CPU clock. The cumulative distribution graph (Figure 6) of the RTTs shows that 90% of the RTTs are below 140 µs for the 200MHz CPU clock, rising to 155µs for 100MHz case, compared with 85µs for the case without the bridge. As can be seen the latency introduced by the bridge is small enough not to be considered a major problem. No packet loss was reported in these tests.

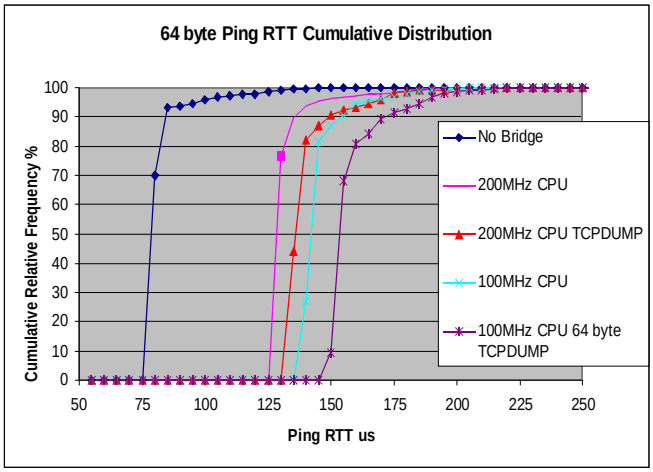

Figure 6 Ping RTT Cumulative Distribution

A test was also conducted with tcpdump capturing 30 000 packets crossing the bridge. The traces were written to a file. In this case the average latency increased from 24.5 to 28.5µs with a 200MHz CPU clock and from 32 to 38µs for the 100MHz case. We can conclude that running tcpdump on the bridge to capture packet traces does not increase the bridge latency significantly. Again, no packet loss was reported.

The tests were repeated for a 1480 byte ping packets; the results are given in Table 4. The average latency in this case was 144.5µs with a 200MHz CPU clock and 153.5µs for the 100MHz case. This increase is expected as the bridge must process a longer packet.

Comments	*CPU Clock MHz*	*Min ms*	*Average ms*	*Max ms*	*Std Dev*	*Bridge Latency µs*
No Bridge	NA	0.322	0.334	2.146	0.013	0
Bridge	100	0.625	0.641	3.091	0.017	153.5
Bridge	200	0.608	0.623	3.169	0.016	144.5

Table 4 Ping RTT and Bridge Latency 1480 Byte Pings

Figure 7 Shows the cumulative distribution of RTTs. In this case 90% of the RTTs are below 630 µs for the 200MHz CPU clock, rising to 655 µs for 100MHz case, compared with 340 µs without the bridge.

Running tcpdump on the bridge increases the average latency to 146µs (200MHz) and 158.5µs (100MHz). Again the increase due to tcpdump is small.

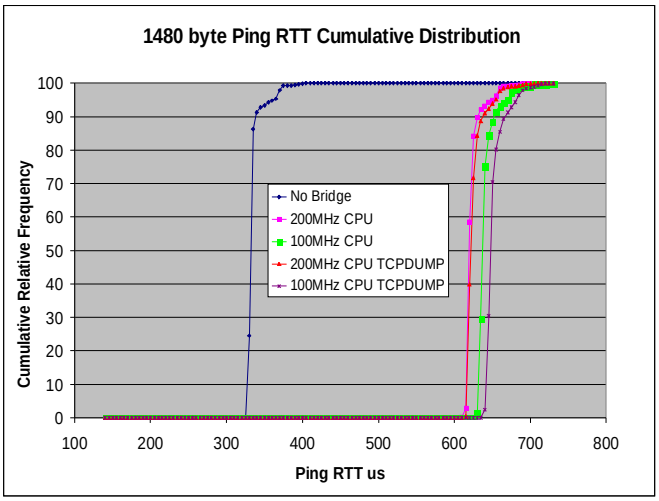

Figure 7 Ping RTT Cumulative Distribution

The above tests were conducted at a packet rate of 50pps (ie 20 ms spaced packets). To increase the ping packet rate to 500pps we changed the kernel granularity on the FreeBSD machine to 1 ms and repeated the tests. (Figure 8). The granularity of the bridge was not changed from 100Hz. See Table 5 and Table 6 for a summary of the results. One ping packet was lost in the case of the 1480 byte ping test with 100MHz CPU clock.

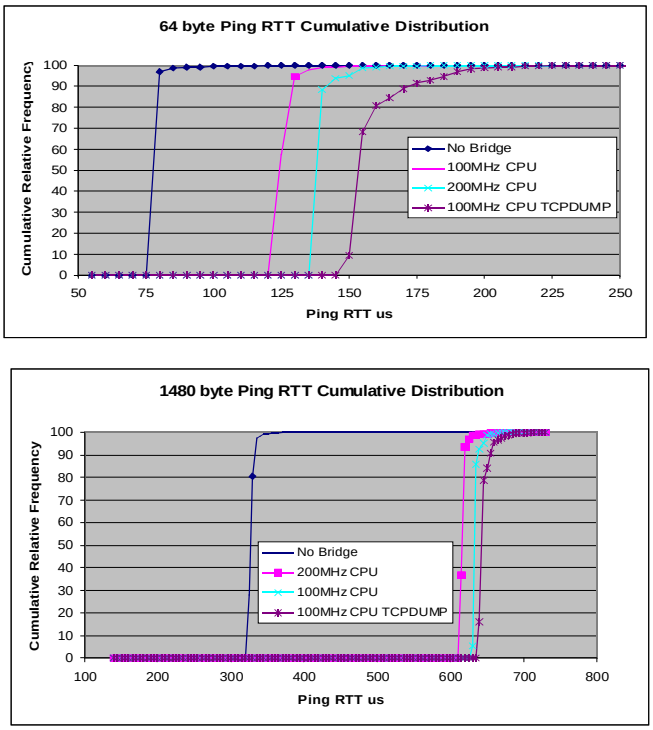

Figure 8 500pps Ping RTT Cumulative Distribution

Comments	*CPU Clock MHz*	*Min ms*	*Average ms*	*Max ms*	*Std Dev*	*Bridge Latency µs*
No Bridge	NA	0.073	0.079	2.341	0.010	0
Bridge	100	0.132	0.139	2.524	0.010	30
Bridge	200	0.122	0.126	2.336	0.01	23.5

Table 5 Ping RTT and Bridge Latency 64 Byte Packet

Comments	*CPU Clock MHz*	*Min ms*	*Average ms*	*Max ms*	*Std Dev*	*Bridge Latency µs*
No Bridge	NA	0.32	0.33	2.01	0.01	0
Bridge	100	0.62	0.63	2.74	0.01	153
Bridge	200	0.61	0.62	2.35	0.01	144.5

Table 6 Ping RTT and Bridge Latency 1480 Byte Packet

The average latency for the higher packet rates is similar to that measured in the 50pps case. We can conclude that the bridge is suitable for capturing packets at this rate.

Running tcpdump on the bridge increases average the latency from 30µs to 36µs (100MHz) for a 64 byte ping. In the case of the 1480 byte ping the average latency increases from 139µs to 158.5µs (100MHz). Again the increase due to tcpdump is small and is similar to the 50pps case. The test was not conducted at 200MHz as the results for the 100MHz clock speed are similar to the 50pps case.

F. Hardware Issues

The original configuration of the bridge used the built in adapter cs1, a 10BaseT adapter (CS8920 Ethernet Adapter), card and a 10/100BaseTX D-Link DFE-530TX (VT6102 Rhine II) adapter. The bridge did not work using this configuration. The problem was fixed by installing a second D-Link network interface card and using the two D-Link cards as the bridge ports.

With this configuration we observed some unusual behaviour in the RTTs (outlined further in Appendix 3). We solved the problem by replacing the D-Link Ethernet adapters with SMC types.

IV. DUMMYNET

To use the bridge as a traffic shaper dummynet needs to be installed. This can be achieved by either including the ip firewall (ipfw) and dummynet options and rebuilding the kernel, or by loading the appropriate modules dynamically using “kldload”.

Dummynet uses the ipfw code to select packets that are to be subjected to packet delay, random loss or bandwidth limitation. The selected packets are extracted from a bounded size queue at a programmed rate and passed to a second queue where the delay, if any, is added. Packets leaving the second queue are re-injected into the protocol stack at the same point they came from. Details of dummynet rules and commands are given in [2] and the dummynet man page.

Again, the ping method was used to measure the latency that the ipfw and dummynet introduce across the bridge. Dummynet performs its task once per timer-tick; therefore, the kernel granularity (HZ option) was changed from 100Hz to 1000Hz so that packets could be delayed in increments down to 1 ms.

A. Dummynet Results

The round-trip times in ms and bridge latency with the dummynet delay set to 0ms (ipfw pipe 1 config delay 0ms) were as follows (Table 7):

Comments	***CPU Clock MHz***	***Min ms***	***Average ms***	***Max ms***	***Std Dev***	***Bridge Latency*** µs
No Bridge	NA	0.077	0.082	2.660	0.015	0
Dummynet	100	0.171	0.19	2.227	0.017	51.5
Dummynet	200	0.150	0.159	1.283	0.015	38.5

Table 7 Ping RTT and Bridge Latency with ipfw and Dummynet Installed

Installation of the firewall for dummynet support increases the average latency from 24.5µs to 38.5µs (200 MHz CPU clock) and from 32µs to 51.5µs (100 MHz CPU clock). The maximum RTT reported by the ping test was 1.283 ms, this value only occurred once. Figure 9 is a plot of the cumulative distribution of the RTTs with dummynet installed. We can see that 90% of the ping RTTs are below approximately 170 µs (200 MHz CPU clock) and 200 µs (100 MHz CPU clock) and 99% of the RTTs are less than 205 µs (200 MHz CPU clock) and 240 µs (100 MHz CPU clock).

A simple test was conducted where the latency of the

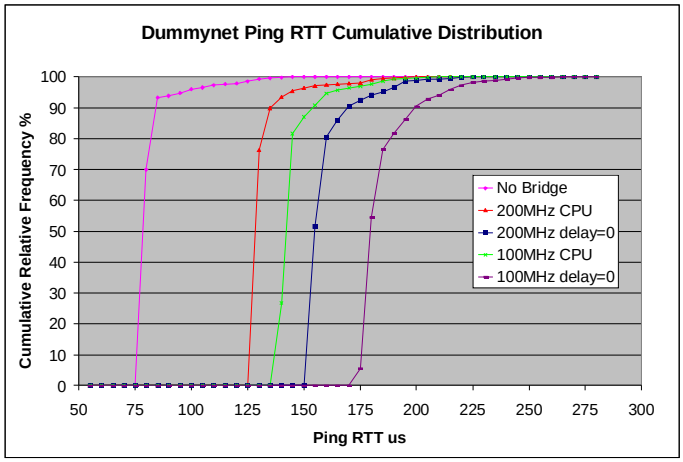

Figure 9 RTT Cumulative Distribution with Dummynet Installed

bridge was varied from 1000ms to 1ms, and ping packets where sent from the FreeBSD PC to the WinPC. The delay was varied by the command: ipfw pipe 1 config delay "delay"ms. In this test case "delay" was changed from 1000 to 1 ms. The results are summarised in Table 8.

Dummynet Delay ms	***Minimum RTT ms***	***Average RTT ms***	***Maximum RTT ms***	***Standard Deviation***	Average Oneway Delay
1.00	0.93	1.57	3.86	0.29	0.78
2.00	3.02	3.56	4.26	0.29	1.78
5.00	9.06	9.56	10.24	0.28	4.78
10.00	18.84	19.75	20.34	0.29	9.88
100.00	198.88	199.59	200.51	0.30	99.80
1000.00	1998.77	1999.37	1999.88	0.29	999.69

Table 8 Dummynet Delay Statistics

The ip firewall uses the sysctl variable "net.inet.ip.fw.one_pass", to determine if the packets coming from a pipe can be either directly forwarded to their destination, or passed again through the ipfw rules. In this test case the "net.inet.ip.fw.one_pass" variable was set to 1, thus each packet was only processed (hence delayed) once in each direction. As can be seen the RTT is approximately double the delay introduced by the

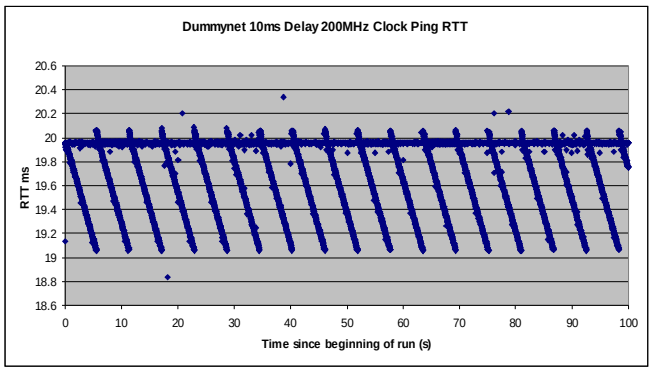

Figure 10 RTT with 10ms Delay

bridge.

Figure 10 is a typical RTT plot, we can see that the delay through the bridge is not constant, but changes periodically. The difference between the maximum delay and the minimum delay is 1 ms (kernel granularity[3]). This behaviour is consistent with dummynet performing its task once per time tick (1ms). Packets that arrive between timer ticks will miss one tick's worth of delay.

Figure 11 shows the cumulative frequency distributions for delays up to 5 ms. It is interesting to note that most of the actual delays are less than the specified delay.

3 The range becomes 10ms for a kernel granularity of 10ms.

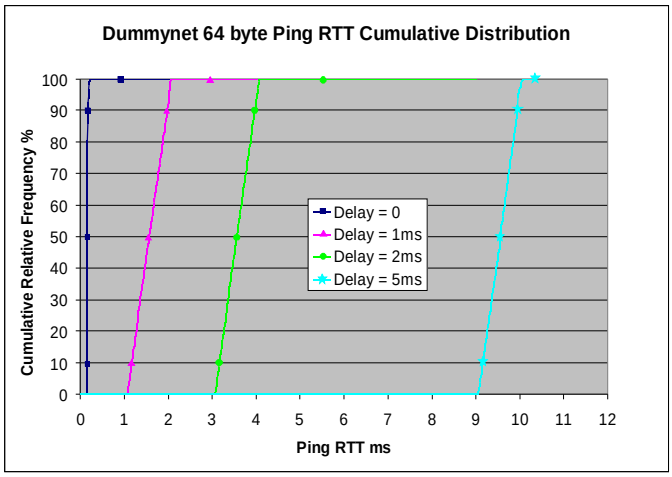

Figure 11 RTT Cumulative Distribution

V. Conclusion

An inexpensive Ethernet Bridge can be built from obsolete Pentium PCs and Ethernet cards. The time-stamping accuracy of our specific bridge based on an IBM PC 300GL is satisfactory for packet tracing where extremely short packet interarrival times are not expected.

The interval between ping packets is related to the kernel granularity; with the default granularity of FreeBSD 4.6 the interval is a multiple of 10ms. Smaller intervals between ping packets can be obtained by increasing the kernel HZ option.

The RTT of 64 byte ping packets was used to obtain an estimate of the average latency across the bridge. Using this method, we found that the average bridge latency was 32 µs for a 100MHZ CPU clock and 24.5 µs for a 200MHZ CPU clock. For large packets (1480 bytes) the latency increases to 144.5µs with a 200MHz CPU clock and 153.5µs for the 100MHz case. Running tcpdump on the bridge increases the bridge latency by only 4µs for a 64 byte packet with a 200MHz CPU clock and 6µs for a 100MHz CPU clock.

The bridge is also useful as a traffic shaper by installing dummynet and ipfw. The overhead introduced by installing ipfw and dummynet increased the average latency from 24 to 37 µs for a 64 byte packet.

This technical report outlined a simple procedure for estimating the bridge latency using ping packets and PCs. A more accurate estimate of the latency can be obtained by using appropriate test equipment and the methodology outlined in RFC1944. The results obtained with this simple method should be compared with those that would be obtained from RFC1944. Further work also needs to be done to characterise the packet loss performance of the bridge. The results obtained with the Pentium based bridge should also be compared to a bridge based on a more powerful processor to determine if this significantly decreases the latency. These issues will be addressed in a subsequent report.

Although a Pentium processor based PC was used in this case it would be interesting to try a 486 processor machine as the basis of the bridge.

If only dummynet is required then it may be possible to run the bridge without a hard disk by using a single floppy disk operating system such as picoBSD [5].

Acknowledgments

I am grateful for the assistance that Grenville Armitage gave me in preparing this report.

References

[1] S. Peterson “FreeBSD Handbook Chapter 19 Advanced Networking” http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/bridging.html"

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

[3] The Free BSD Project “FreeBSD/i386 4.6.2-RELEASE Installation Instructions”

[4] The Free BSD Project “FreeBSD Handbook Sec 2.2.3.1 Disk Layouts for the i386"

[5] “PicoBSD” http://people.freebsd.org/~picobsd/picobsd.html

Appendix 1

Bridge Hardware Description

A. PC:

IBM PC 300GL Pentium 166MMX type 6282-36A S/N 90-A3HD5

64MB RAM

256kB Cache

1704MB IDE HDD

Inbuilt VGA, USB, 10Mbps Ethernet, 3.5" floppy

The CPU multiplier and the bus speed can be changed by setting a dip-switch on the motherboard. The range of available CPU clock frequencies is 75MHz to 233MHz the processor speed is reported in the System Summary of the Bios.

B. Ethernet network interface cards:

Two PCI Fast Ethernet NICs were added details as follows:

tx0: SMC EtherPower II 10/100
Mac address: 00:e0:29:2e:bd:b6,
type SMC9432TX

tx1: SMC EtherPower II 10/100
Mac address: 00:e0:29:2e:c7:d0,
type SMC9432TX

C. Operating System:

FreeBSD 4.6 Release 0.

D. Dmesg Output

dmesg reports the following for a CPU speed of 166 MHz:

```
Copyright (c) 1992-2002 The FreeBSD Project.
Copyright (c) 1979, 1980, 1983, 1986, 1988, 1989, 1991, 1992, 1993, 1994
The Regents of the University of California. All rights reserved.
FreeBSD 4.6-RELEASE #0: Thu Sep  5 11:47:41 EST 2002
root@Sniffer1.caia.swin.edu.au:/usr/src/sys/compile/MYKERNEL
Timecounter "i8254" frequency 1193182 Hz
CPU: Pentium/P55C (166.19-MHz 586-class CPU)
Origin = "GenuineIntel" Id = 0x543 Stepping = 3
Features=0x8001bf<FPU,VME,DE,PSE,TSC,MSR,MCE,CX8,MMX>
real memory = 67108864 (65536K bytes)
avail memory = 60403712 (58988K bytes)
Intel Pentium detected, installing workaround for F00F bug
md0: Malloc disk
npx0: <math processor> on motherboard
npx0: INT 16 interface
pcib0: <Host to PCI bridge> on motherboard
pci0: <PCI bus> on pcib0
isab0: <Intel 82371SB PCI to ISA bridge> at device 1.0 on pci0
isa0: <ISA bus> on isab0
atapci0: <Intel PIIX3 ATA controller> port 0xfff0-0xffff at device 1.1 on pci0
ata0: at 0x1f0 irq 14 on atapci0
ata1: at 0x170 irq 15 on atapci0
uhci0: <Intel 82371SB (PIIX3) USB controller> port 0x5800-0x581f irq 11 at device 1.2 on pci0
usb0: <Intel 82371SB (PIIX3) USB controller> on uhci0
usb0: USB revision 1.0
uhub0: Intel UHCI root hub, class 9/0, rev 1.00/1.00, addr 1
uhub0: 2 ports with 2 removable, self powered
pci0: <Cirrus Logic GD5446 SVGA controller> at 8.0
tx0: <SMC EtherPower II 10/100> port 0x5400-0x54ff mem 0x60001000-0x60001fff irq 10 at device 10.0 on pci0
miibus0: <MII bus> on tx0
qsphy0: <QS6612 10/100 media interface> on miibus0
qsphy0: 10baseT, 10baseT-FDX, 100baseTX, 100baseTX-FDX, auto
tx0: address 00:e0:29:2e:bd:b6, type SMC9432TX
tx1: <SMC EtherPower II 10/100> port 0x5000-0x50ff mem 0x60000000-0x60000fff irq 11 at device 11.0 on pci0
miibus1: <MII bus> on tx1
qsphy1: <QS6612 10/100 media interface> on miibus1
qsphy1:10baseT, 10baseT-FDX, 100baseTX, 100baseTX-FDX, auto
tx1: address 00:e0:29:2e:c7:d0, type SMC9432TX
orm0: <Option ROM> at iomem 0xc0000-0xc7fff on isa0
fdc0: <NEC 72065B or clone> at port 0x3f0-0x3f5,0x3f7 irq 6 drq 2 on isa0
fdc0: FIFO enabled, 8 bytes threshold
fd0: <1440-KB 3.5" drive> on fdc0 drive 0
atkbdc0: <Keyboard controller (i8042)> at port 0x60,0x64 on isa0
atkbd0: <AT Keyboard> flags 0x1 irq 1 on atkbdc0
kbd0 at atkbd0
vga0: <Generic ISA VGA> at port 0x3c0-0x3df iomem 0xa0000-0xbffff on isa0
sc0: <System console> at flags 0x100 on isa0
sc0: VGA <16 virtual consoles, flags=0x300>
sio0 at port 0x3f8-0x3ff irq 4 flags 0x10 on isa0
sio0: type 16550A
sio1 at port 0x2f8-0x2ff irq 3 on isa0
sio1: type 16550A
ppc0: cannot reserve I/O port range
cs1: <CS8920 Ethernet Adapter> at port 0x250-0x26f irq 5 on isa0
BRIDGE 020214 loaded
ad0: 1625MB <ST31722A> [3303/16/63] at ata0-master WDMA2
Mounting root from ufs:ad0s1a
```

Appendix 2

A. Bridge Software Configuration

Outlined below are the configuration changes required to set up the bridge, ipfw and dummynet. A shell script "bridge.sh" that automatically installs bridging, ipfw and dummynet is available from the author.

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

```
bridge_load=\"YES\"    # load bridge module
```

- For dummynet must add the ipfw module

```
ipfw_load=\"YES\"      # load ipfw module
```

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.

a) Create a shell script in /usr/local/etc/rc.d/ with the following line:

```
kldload dummynet
```

b) Make sure that the shell script is executable.

Alternatively rebuild the kernel with the options specified in the FreeBSD Handbook section on bridging (http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/bridging.html) and the dummynet man page. To recompile kernel refer to http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/kernelconfig-building.html

- Dummynet performs its task once per timer tick; since the default granularity is 100 Hz, this limits delays to be larger than 10 ms. For shorter than 10 ms delays, one needs to increase the granularity.

```
options HZ=1000        #sets the timer granularity to 1000Hz
```

- Add the following lines 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=tx0:1,tx1:1
```

To enable ip firewall:

```
net.link.ether.bridge_ipfw=1
#force bridged packets to pass through the firewall code.
```

For dummynet as recommended in dummynet [2]:

```
net.inet.ip.fw.enable=1
#enables firewall in the IP stack
net.inet.ip.fw.one_pass=1
#Forces a single pass through the firewall. If set to 0, packets coming out of a pipe will be reinjected into the firewall starting with the rule after the matching one. NOTE there is always one pass for bridged packets.
```

- I've added these lines to /etc/rc.conf

```
gateway_enable="YES"    # probably already there
firewall_enable="YES"
firewall_script="/etc/rc.ipfw.dummynet"
#Name of script with firewall commands
```

The following lines are used to control the delay

(this was put in /etc/rc.ipfw.dummynet, but it can be issued from the command line).

```
ipfw add pipe 1 ip from any to any bridged
ipfw pipe 1 config delay "delay in ms"
```

Appendix 3

A. Ethernet Network Interface Card Issues

The original configuration of the bridge used the built in adapter cs1, a 10BaseT adapter (CS8920 Ethernet Adapter), card and a 10/100BaseTX D-Link DFE-530TX (VT6102 Rhine II) adapter. The bridge did not work using this configuration. The problem was fixed by installing a second D-Link network interface card and using the two D-Link cards as the bridge ports. I haven't been able to establish why the bridge did not work with the two different cards.

With the two D-Link cards used for bridging (vr0 and vr1) and using ping to measure the latency uncovered some unusual behaviour. Although the average RTT time was low both the variance and maximum RTT were high(Table A1). The high standard deviation suggested that there were many packets that were delayed considerably more than the average.

The round-trip times in ms were as follows:

	CPU Clock (MHz)	*Number of pings*	*Min*	*Average*	*Max*	*Standard Deviation*
No Bridge	NA	30000	0.083	0.109	2.147	0.024
Bridge	100	15000	0.176	0.227	2.150	0.065
Bridge	200	40000	0.165	0.231	3.823	0.132

Table 9 Ping RTT

Note: The same experimental setup as outlined in the main body of the report was used.

Scatter plots were used to visually check the delays, from these plots some "interesting" behaviour was observed.

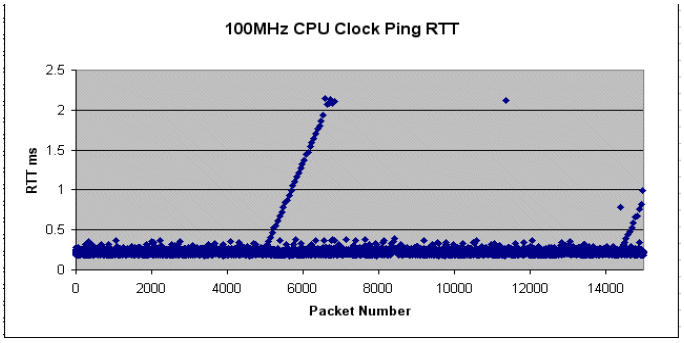

Figure 12 Ping RTT with the bridge between PCs, Bridge CPU clock 100MHz

Figure 12 is a plot of ping time vs ping packet number across the bridge. We see that there is a gradual ramping up of the RTT for ping packets numbered between about 5000 and 7000, this feature repeats itself starting from packet 14500 approximately. The behaviour is periodic with a period of approximately 90-95 seconds.

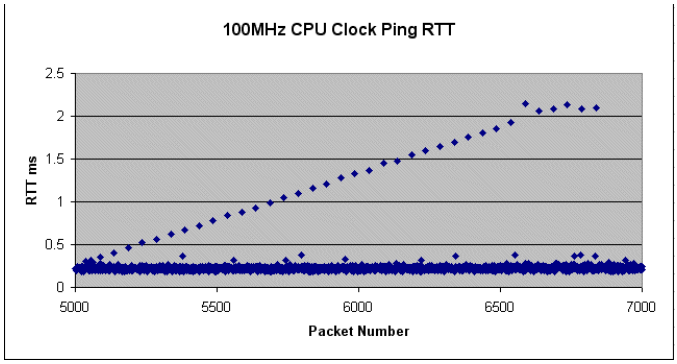

Figure 13 Ping RTT for packets 5000 to 7000

Closer inspection of the ramp reveals that every 50th packet is delayed, see Figure 13. Although this behaviour may not be an issue for a PC connected to a LAN, it is undesirable when using the PC to capture packet traces.

The clock speed of the processor was raised to 200MHz and the experiment repeated, again the behaviour is similar as it is for the 100 MHz case. Refer to Figure 14 below, (note Excel can only display 32000 points):

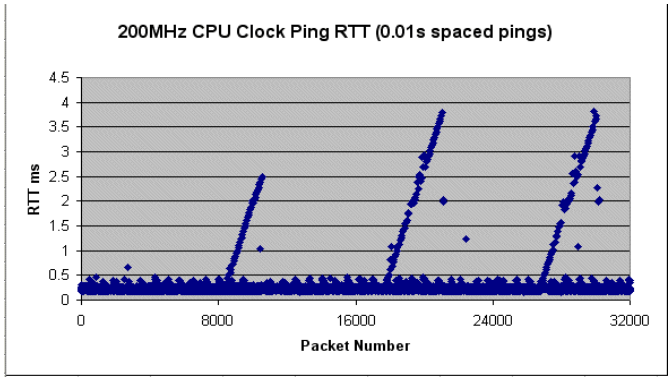

Figure 14 Ping RTT with the bridge between PCs, Bridge CPU clock 200MHz

Tcpdump packet traces confirmed that the large delays were introduced by the bridge. Due to the periodic nature of the delays, it was initially assumed that some process running in BSD or some interrupt on the motherboard was causing the increase in latency. Shutting down processes did not affect the behaviour, this pointed to either one of the key FreeBSD daemons that could not be shutdown (eg inetd), or a hardware related issue.

The Ethernet cards were changed to SMC EtherPower II 10/100 and the ramping of delay disappeared. The behaviour reappears if one D-Link and one SMC are installed, suggesting that the D-Link card is the culprit. In summary, when choosing Ethernet cards for building devices used to measure interarrival times, one must be careful to ensure that the particular network interface cards chosen do not have any unusual effects on the packet delays introduced by the bridge.