A Measurement Study of Packet Reception using Linux

Michael J. Schultz, mjschultz@gmail.com (A paper written under the guidance of Prof. Raj Jain)

Download

Abstract

Packet capturing is an important part of a network administrator's tool-chain. It is often used to "dig down" into the specific traffic traversing a network. Because of this, it is important that a packet capturing method provides a correct view of how the network acts. Thus, a packet capturing platform must be able to keep up with the line-rate of the router it is co-located with; for most enterprise level networks this is 1 gigabit per second.

The purpose of this measurement study is to determine what software packet capture method works best and what hardware platform is able to support the speeds. This is done through two two-factor full factorial experimental designs which test three packet capturing methods under two kernel configurations and three packet capturing methods on three machine types. The results indicate that the kernel configuration does not have as great an effect as previously thought when compared to the packet capture method; they also show that 10 GbE systems have a long way to go towards efficient use of resources in user-space.

Keywords: Packet Reception, Packet Capturing Methods, High-Speed Networking, Performance Analysis, Experimental Design, Linux, Kernel Module, Resource Utilization

Table of Contents

1 Introduction

Network administrators frequently use packet capturing to get a glimpse into how their network is being used. This information can then be used to analyze the data to detect network threats, make operations decisions, or troubleshoot network problems. No matter what the network administrators need to do, an enterprise that wants to know how their network behaves would install a packet capturing device near their routers. These routers are able to handle aggregate traffic rates that typically run at 1 gigabit per second (Gbps). With this being the case, it is important that packet capturing is able to achieve similar rates to get the most comprehensive view of the network.

1.1 Motivation

With aggregate speeds approaching 1 Gbps, most systems have a hard time keeping up with the packet flow. Recently, there have been suggested changes to the software stack that allow the software to keep closer pace with the networking hardware. While hardware specific approaches to packet capture have been built and successfully sold in the past, these are typically more expensive and less flexible than software approaches. This case study looks at different approaches to software packet capture and evaluates their performance under different configurations to help quantify what matters the most.

It should also be noted that this work has been developed using resources freely available from Washington University’s Open Network Lab [ONL]. This means the results of this study gives software developers and researchers a baseline throughput for machines available for experimenters, if they are configured as in this case study.

1.2 Background

There have been various works in recent years looking at evaluating existing packet capture techniques [Mrazek08][Braun10]. Both these papers are excellent case studies and should compliment the results of this study well. Another approach looks at capturing 10 Gbps traffic by splitting it up into 10 separate 1 Gbps streams [Schneider07], though it is not clear their machines would be able to keep up with a continuous 1 Gbps stream.

An interesting idea that is able to capture at high-speeds is one that lies between using a full O/S like Linux and purchasing dedicated hardware: a special purpose kernel [Huang10]. This avoids the heavy cost of special hardware by pushing all the available resources to the network card and CPU. It also comes with the burden of debugging in an unfamiliar environment and developing software against a different programming interface.

There are also several hardware-based solutions that can be purchased from vendors that claim to be able to sustain 1 Gbps packet capture [Endace][Napatech][WildPackets]. Aspects of these hardware solutions have also trickled into the commodity gigabit Ethernet adapters that are available today such as multiple receive queues [Intel06].

1.3 Linux Kernel

The Linux kernel has been in active development for 20 years and has a strong support community and many features. One of the key features relating to packet capture is the “New Application Programming Interface” (NAPI) [Salim01]. NAPI was created when it was observed that under heavy packet load the Linux kernel enters a “receive livelock” state in which the processor is continually interrupted and cannot make useful progress [Mogul97]. To avoid this issue a driver designed against the NAPI will insert a job on a processor queue for a network interface and that interface will not generate more interrupts until the job has executed. Packets will continue arriving and be buffered in memory and once the job begins executing it will poll the network interface for more packets until there are none left. Thus, when the network interface is lightly loaded Linux can process packets normally and when the interface is heavily loaded the kernel is able to make progress in packet processing. The next section discusses the different packet capture methods available on a modern Linux kernel.

2 Packet Capture Methods

Packet capturing is important to network administrators to help identify security problems, troubleshooting errors, and benchmarking. Several methods of packet capturing have been proposed. Figure 1 shows the three software packet reception paths that are discussed below (from left to right): handling packet capture in the kernel, using the default `PF_PACKET` code path, and using the improved `PF_RING` patch. There is also discussion of more costly hardware options.

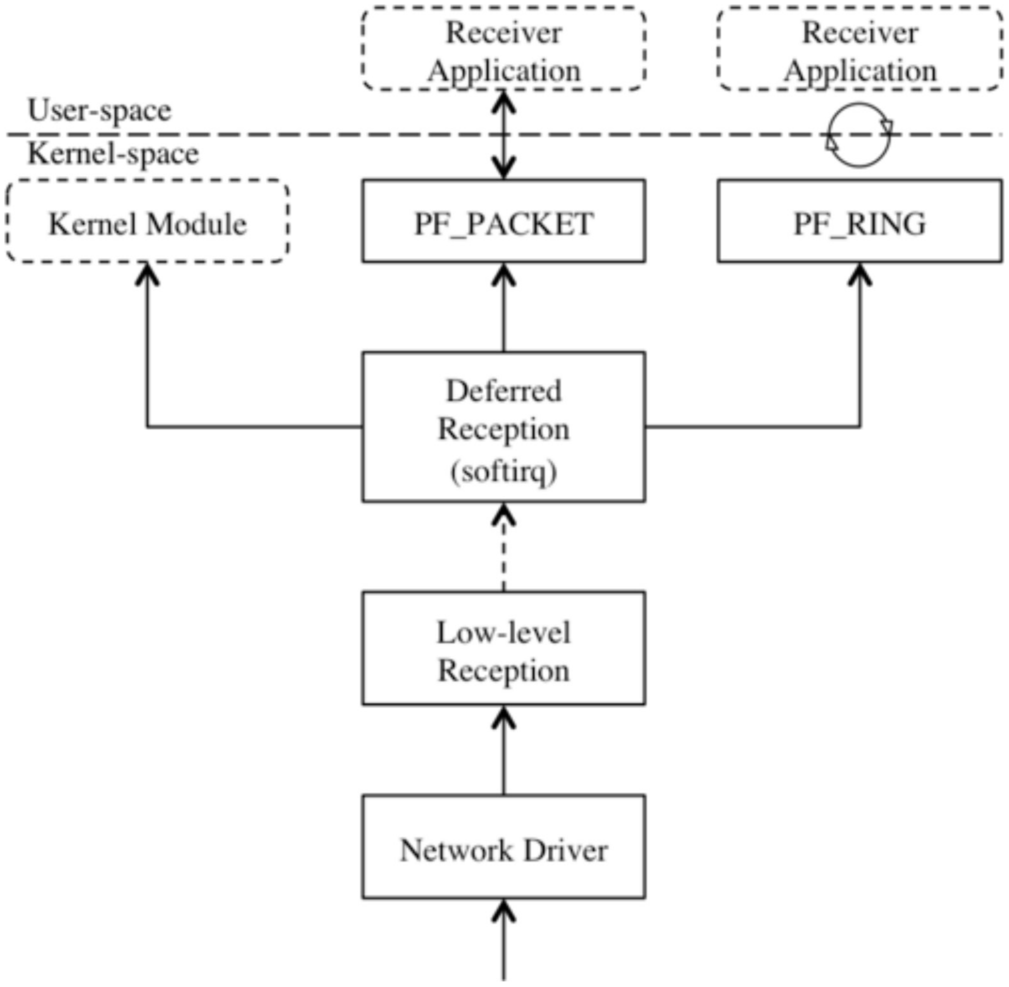

Figure 1: Three packet reception paths in the Linux kernel.

2.1 Kernel Module

The Linux kernel allows loading new code into the kernel, allowing a programmer to avoid costly interactions between user-code and kernel-code [Jones08][Birch10]. These pieces of code that can be loaded into the kernel are called “kernel modules” and can hook into various parts of the executing kernel. This lets a developer grab a received packet early and handle it before much time is spent processing in the network stack. Once the kernel module has handled the packet it can return it to the Linux kernel for further processing or discard the packet so no more time is spent on it. By using a kernel module, packet capturing can happen almost immediately instead of spending cycles traversing the network stack, moving into user-space, and completing the capture process [Birch10].

However, kernel module development is more costly than normal development. A kernel developer must understand the internals of Linux before writing the module. If there are errors in the module the entire system can become unstable and halt. Over time the kernel internals may also change, making the module out-of-date and locking it to a fixed kernel version. A normal user-space program will simply display an error message in the worst case, or be able to recover and continue in the best case. Thus, using a kernel module for packet capture can be a risky endeavour but performance improvements may be worthwhile. A safer, less efficient user-space packet capturing method is presented next.

2.2 `PF_PACKET`

A standard Linux kernel ships with the `PF_PACKET` protocol family that supports a user-space program interacting directly with a packet instead of allowing Linux to handle IP processing. Normally when a user opens a network socket the packet is received by the kernel, pushed through the IP receive function then through the TCP receive function and finally the encapsulated data is given to the user program. `PF_PACKET` instead makes a copy of the packet buffer before the IP receive function is called and sends the entire packet to the user program. This causes the system to skip the normal packet reception path and allows a user-space program direct access to the contents of every packet.

Unlike the kernel module approach, writing programs in user-space is more robust but comes with the performance cost of crossing the kernel-space/user-space boundary. A packet capture program written for user-space is portable between different versions of the same operating system and likely portable between completely different operating systems—leading to a much shorter development investment. Crossing the kernel-space/user-space boundary typically happens through the system call interface which has significant system overhead. `PF_RING` attempts to relieve these drawbacks and is discussed next.

2.3 `PF_RING`

A newer approach comes from Luca Deri's `PF_RING` protocol family which acts as an alternative to `PF_PACKET` with a different approach [Deri04][Fusco10]. Recognizing the cost of the per-packet system calls, `PF_RING` creates a ring buffer in the kernel and uses a kernel/user memory map to share that buffer with the user-space program. In the beginning, the user-space program does one system call asking for packets; if there are packets it begins processing on the ring until there are no more packets available at which point it will perform a second system call which will not return until there are more packets to process. This allows the user-space program to spend more time processing packets than working through system call overhead.

Though `PF_RING` provides a simple option for high-speed packet capture in user-space, it requires a kernel module to provide the protocol family, some small modifications to existing user-space programs and it is currently tied to Linux. Recent versions of `PF_RING` only require the kernel module to be compiled against the running kernel and no longer requires the kernel to be specially patched to support the software. A modified version of `libpcap` (a standard packet capture library) is available to existing applications to transparently use `PF_RING` instead of `PF_PACKET`. So, while `PF_RING` provides support for user-space applications to perform high-speed packet capture it comes at the cost of being tied to a kernel module which is not guaranteed to be kept up-to-date as the kernel version progresses.

2.4 Hardware Options

As an alternative to the software approaches above, there are hardware based products from Endace, WildPackets, and Napatech that offer high-speed packet capture devices [Endace][Napatech][WildPackets]. These devices can be quite costly when compared to a normal network card and can also limit the functionality that a user may want from the device. Unfortunately, due to the cost of these devices, it is not possible to test any of the hardware options against the software methods for this study.

The performance evaluation of the three software-based packet capture methods discussed at the beginning of this section is presented in the next section.

3 Performance Evaluation

To evaluate the performance of packet capturing a measurement study is performed. Given the complexity of the system there are several parameters that can vary, but these can be reduced to a more manageable number of factors. These factors help find the maximum throughput of the Linux kernel under several conditions. Finally, the results of the experiments are presented and analyzed, and a brief discussion of their meaning concludes this section.

3.1 Performance Factors and Goals

There are many parameters that can affect the packet capture performance of a system. Below is a list of the most important parameters.

- Network card line-rate of receiver
- Processor type of receiver
- Memory bandwidth of receiver
- Kernel version of receiver
- Kernel configuration of receiver
- Network card driver of receiver
- Capturing software of receiver
- System load of receiver
- Packet size and rate

The network card installed on the receiver can cause major differences (a 10 megabit per second Ethernet card simply cannot keep pace with a 10 gigabit per second Ethernet card) or a minor difference (two 1 gigabit per second Ethernet cards by different manufacturers). The processor speed and architecture can change the number of instructions executed in the time it takes to receive a packet. The speed at which the network card and processor can access memory to write and read the packet also affects performance at the hardware level.

On the software side of packet processing, the version of the Linux kernel can impact performance as it is constantly developing. Similarly the way the kernel is configured (what pieces of code are included) can cause unnecessary overhead slowing down the system. The hardware driver may have bugs or inefficiencies that can change as well. And of course the variety of packet capture options will affect performance in different ways.

External events that are even more difficult to control and affect the performance include the system load of the packet receiver and the size and rate at which packets arrive at the network card. Depending on the type of packet a network card is capturing, it can receive over 1.4 million (64 byte) packets per second or as few as 81 thousand (1522 byte) packets per second.

The purpose of this study is to find the packet capture method that results in the highest throughput for a given system. This will be measured in megabits per second (Mbps) and each packet is a minimum sized Ethernet packet that includes the Ethernet preamble (7 bytes), start of frame delimiter (1 byte), Ethernet frame (64 bytes), and the interframe gap (12 bytes). CPU utilization on the receiver will also be recorded.

3.2 Experimental Setup

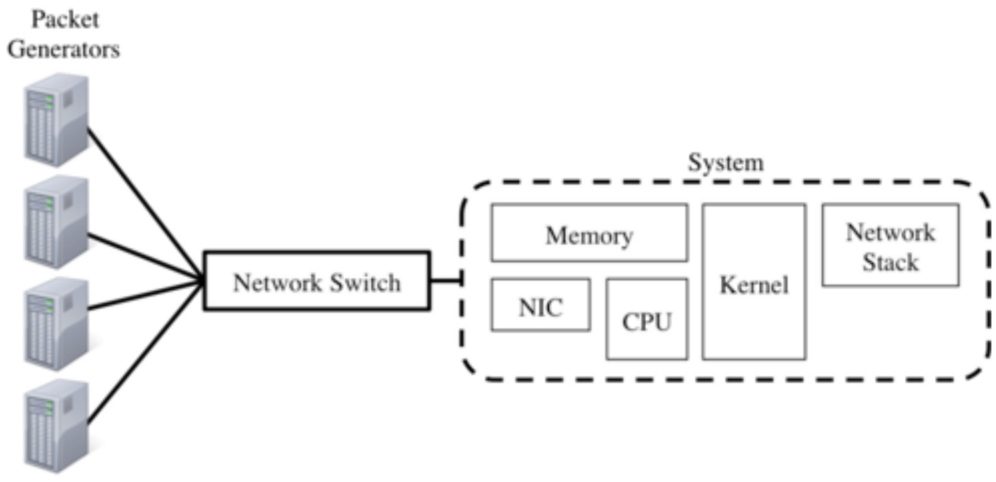

Figure 2: Breakdown of the evaluation setup and system under test.

The general setup of the experiment can be seen in Figure 2 above. There is a collection of packet generators on the right side of the figure, each of these packet generators uses the Linux Kernel Packet Generator (LKPG) [Olsson05]. The LKPG is a kernel module that is loaded and configured at run time and is able to generate packets as fast as the processor is capable. The packets generated are minimum sized Ethernet frames (64 bytes + 20 bytes of Ethernet overhead). It is assumed that the LKPG is not able to generate packets at full line-rate, so multiple instances are started on separate machines each specifying the same receiver. The packet streams from these machines are aggregated at a network switch and sent to the receiver. There is not a mechanism to determine the speed at which the switch is sending packets to the receiver, but it is assumed to be sending at line-rate as the aggregate speed of the packet generators should be greater than line-rate (it is also assumed the processor of a 10 GbE system is unable to keep up with line-rate).

Given the goal of measuring packet capturing throughput, this case study defines the system under test (SUT) to be the packet receiver [Jain91]. In order for a packet to be captured, it must first travel through several pieces of the system shown in Figure 2. First the network card (NIC) receives packet and stores it in memory, then the CPU reads the packet and the kernel begins processing, and finally the packet is given to the network stack to finish the journey.

Throughput is measured at the end of the network stack for each of the capturing software methods. The Kernel module will execute once for every packet and tell the kernel to discard that packet once it has been recorded. The `PF_PACKET` and `PF_RING` capturing methods will run in user-space and receive the packet and will discard the packet after being recorded. This makes the throughput measurement bounded only by the system's ability to deliver the packet to the receiving software and not by actions the receiving software may take.

The receiving application of the kernel module is a kernel module developed to simply count the number of packets and bytes the module receives in a given time frame. `PF_PACKET` will use an application called `pcount` to keep count the number of packets per seconds captures, similarly `PF_RING` uses `pfcount` designed with the same intention. Both `pcount` and `pfcount` provide the number of packets per second. Because each packet arriving is an Ethernet frame of consistent length, the throughput will be computed based on the packet rate of the system. To measure CPU utilization, the `sar` program from the Sysstat performance monitoring tools package [Sysstat] is used.

Table 1: Factors and Levels for Packet Capture Study

Symbol	Factor	Level 1	Level 2	Level 3
A	Kernel Configuration	Default (Shipped)	Custom	
B	Capturing Software	Kernel Module	`PF_PACKET`	`PF_RING`
C	Machine Type	1 GbE Tolapai	1 GbE Nehalem	10 GbE Nehalem

The experiment will be broken into two two-factor full factorial design using the kernel configuration (A) and capturing software (B) in the first experiment and the capturing software (B) and machine type (C) in the second. The machine type and capturing software have three levels, while the kernel configuration only has two, as seen in Table 1. The custom kernel configuration is based on the default configuration with USB, sound, IPv6, and other unnecessary services removed [Gasparakis10].

3.3 Software and Hardware Details

It is important to note that both hardware and software advance at an incredible rate. This means that a case study like this only represents a snapshot of reality and the details of the software and hardware become important. The machine types and their associated specifications are listed below.

1 GbE Tolapai
 Processor: Intel EP80579 Integrated Processor ("Tolapai," 1-core, 32-bit, 256 KiB cache, 1.2 GHz)
 Memory: 1 GiB (DDR2, 800 MHz)
 Kernel: Linux 2.6.18
 Network Card: Intel Tolapai Gigabit Ethernet (controller embedded on chip, `iegbe` driver)
1 GbE Nehalem
 Processor: 2x Intel Xeon L5520 ("Nehalem," 4-core, 64-bit, 8 MiB cache, 2.26 GHz, HyperThreading disabled)
 Memory: 12 GiB (DDR3, 1066 MHz, Registered ECC, 6 GiB/CPU)
 Kernel: Linux 2.6.37
 Network Card: Intel 82576 Gigabit Ethernet (connected via PCI Express 2.0 x4, `igbe` driver)
10 GbE Nehalem
 Processor: 2x Intel Xeon E5520 ("Nehalem," 64-bit, 8 MiB cache, 2.26 GHz, HyperThreading enabled)
 Memory: 12 GiB (DDR3, 1066 MHz, Registered ECC, 6 GiB/CPU)
 Kernel: Linux 2.6.24.7
 Network Card: Intel 82599EB 10 Gigabit Ethernet (connected via PCI Express 2.0 x8, `ixgbe` driver)

The network card driver is the version shipped with the kernel and designed for the network card corresponding to the machine type specification and will not change between configurations. With everything in place, the first experiment determines the relevance of the kernel configuration when compared to the packet capture method and is presented next.

3.4 Kernel Configuration Results

As a time saving measure the two kernel configurations were only tested on the 1 GbE Nehalem system. Measurements are taken after a short 10 second “warm-up” period and sampled every 10 seconds over a period of 120 seconds giving 12 samples. The results presented in Table 2 are the averaged over all samples.

Table 2: Measured packet capturing throughput performance and CPU utilization for 1 GbE Nehalem

	Capturing Software					
	Kernel Module		`PF_PACKET`		`PF_RING`	
Kernel Configuration	**Tput**	**CPU %**	**Tput**	**CPU %**	**Tput**	**CPU %**
Default	943.67	11.92	562.17	18.91	577.08	12.81
Custom	951.75	12.47	495.89	21.85	747.72	12.84

Since the data does not indicate a large difference between the maximum and minimum values and there is no systematic reason for the errors to be non-normal an additive model is used. It should be noted that the throughput using the `PF_PACKET` method and a custom Linux kernel does worse than with the default kernel, this is likely due to some interactions between kernel- and user-space that changed with the custom kernel, since `PF_RING` tries to minimize those interactions it does not suffer from this.

Table 3: Analysis of Variance (ANOVA) Table for Capture Method and Configuration Effects

Component	Sum of Squares	Percentage of Variation	Degrees of Freedom	Mean Square	F-Computed	F-Table
y	39,006,074.26		72			
y..	36,607,259.69		1			
y-y..	2,398,814.57	100.00%	71			
Capture Method	2,195,829.29	91.54%	2	1,097,914.65	47,689.47	3.23
Configuration	25,288.13	1.05%	1	25,288.13	1,098.43	4.08
Interactions	176,177.69	7.34%	2	88,088.84	3,826.26	3.23
Errors	1,519.46	0.06%	66	23.02		

Running an Analysis of Variance (ANOVA) test on the throughput values shows that an overwhelming 91.54% of the variation is explained by the packet capture method and only 1.05% explained by the configuration. Additionally, 7.34% of the variation was caused by the interaction between the packet capture utility and kernel configuration and less than 0.1% was from experimental error.

A visual test of the error values when compared to the predicted values shows that the relative difference is only about 10 time and the distribution is short-tailed. However, applying a logarithmic transformation maintains the explanation of variation as well. In both models, the strength of the results clearly indicates that, when compared to the packet capturing method, the kernel configuration has minimal impact on the throughput. As such, the focus in the next section is on comparing the machine type against packet capturing methods.

3.5 Machine Type Results

This section presents the results of running a two factor experiment comparing the machine type against the packet capture method. As with the above experiment results are averaged over 10 second windows for 120 seconds (12 samples). Unfortunately, the `PF_RING` capture method could not be measured for either the 1 GbE Tolapai of 10 GbE Nehalem machines due to a non-existent driver and kernel/driver incompatibilities, respectively.

Table 4: Measured logarithmic packet capturing throughput performance and CPU utilization, under the default kernel configuration

	Capturing Software			
	Kernel Module		PF_PACKET	
Machine Type	**Tput**	**CPU %**	**Tput**	**CPU %**
1 GbE Nehalem	2.97	11.92	2.75	18.91
1 GbE Tolapai	2.56	100.00	2.29	100.00
10 GbE Nehalem	3.14	2.07	2.81	63.96

It is important to note that a logarithmic transformation has been applied to the data above. This is not surprising because it compares 1 GbE to 10 GbE and processor speed of the machine type differs by a significant amount as well (1.2 GHz single-core versus 2.6 GHz eight-core machines).

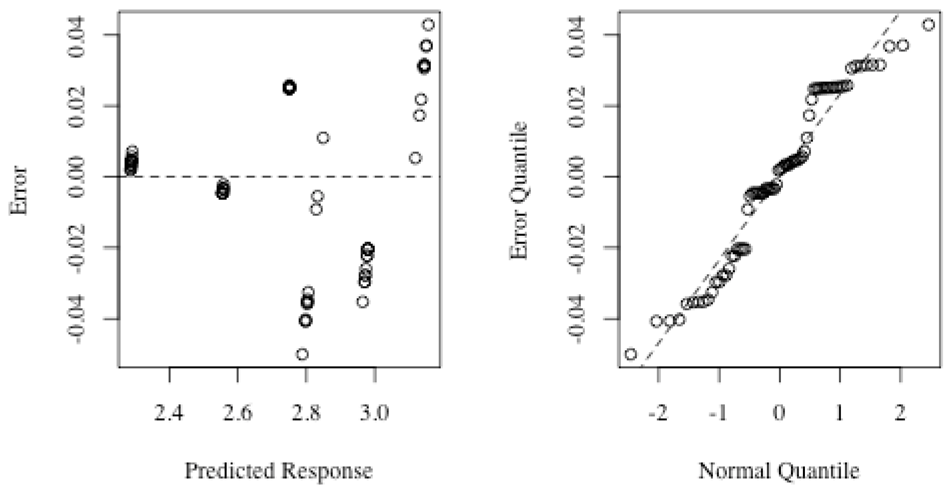

Figure 3: Predicted response versus error values (left) and the quantile-quantile plot (right)

Figure 3 shows the visual analysis for this data. The left graph shows that the data is distributed around 0 and, while there may be outward expansion on the error as the prediction increases the relative magnitude of the error makes that negligible. Similarly, the quantile-quantile plot shows that the errors are normally distributed. This indicates that the logarithmic transformation will create a valid model.

Table 5: ANOVA Table for Logarithm of Machine Type and Capture Method Effects

Component	**Sum of Squares**	**Percentage of Variation**	**Degrees of Freedom**	**Mean Square**	**F-Computed**	**F-Table**
y	551.36		72			
y..	545.85		1			
y-y..	5.51	100.00%	71			
Machine Type	4.11	74.51%	2	2.05	27,391.41	3.23
Capture Method	1.36	24.75%	1	1.36	18,192.99	4.08
Interactions	0.04	0.65%	2	0.02	238.98	3.23
Errors	0.00	0.09%	66	0.00		

Table 5 uses the available data to compare the `PF_PACKET` and kernel module approaches in a second two-factor full factorial experiment. The analysis of variance shows that the machine type causes a 74.51% variation in throughput while the packet capture routine explains 24.75% of the variation. The remaining variation is attributed to interactions between the machine type and packet capture (0.65%) and 0.09% is caused by measurement error.

Because the Nehalem machines are multi-core and multi-threaded while the Tolapai is a single-core system, the average CPU utilization across all cores is may be misleading. It does still speak to the availability of processing resources on a machine type under this type of load if the developer is able to take advantage of the multiple cores. The results above are re-contextualized and summarized in the next section to provide a clear understanding of this work.

4 Summary

This project set out to measure and evaluate several machine types under different kernel configurations with three packet capture methods. Using a two-factor full factorial design, it was determined that the kernel configuration made very little difference when compared to a variety of packet capturing methods. With one factor removed, a second two-factor full factorial design comparing the machine type and capturing software was built. Though not all packet capture methods were used in this experiment, it clearly demonstrates that the machine type has more of an influence than the capturing software. While not the goal of this measurement study, in all experiments the kernel module approach is able to handle significantly more packets than either of the user-space approaches.

Looking beyond the data, the experience with the 10 GbE Nehalem system shows significant room for improvement. This may be caused by using an older kernel version, but the amount of CPU resources used to get packet throughput comparable to a similar 1 GbE Nehalem system is concerning. Future work may look into bringing the 10 GbE Nehalem system up-to-date with the Linux kernel and re-evaluating.

It should also be noted that although `PF_RING` no longer requires kernel patches it does ship with a small number of drivers that are “`PF_RING`-aware” which have dependencies to the kernel version. This is important because `PF_RING` is the only packet capture routine to approach full gigabit speeds [Fusco10]; it still has strong ties to a kernel version just like the kernel module implementation. While it does open up user-space applications for packet capturing, the cost of getting the correct drivers running may offset the advantages.

References

Ordered by relative importance to this paper.

1. [Braun10] Lothar Braun, Alexander Didebulidze, Nils Kammenhuber, and Georg Carle, “Comparing and Improving Current Packet Capturing Solutions based on Commodity Hardware,” Internet Measurement Conference 2010, http://conferences.sigcomm.org/imc/2010/papers/p206.pdf. Presents an analysis of packet caputring on various platforms using different capturing techniques.
2. [Birch10] Samuel W. Birch, “Performance Characteristics of a Kernel-Space Packet Capture Module,” Master’s thesis, Air Force Institute of Technology, Wright-Patterson Air Force Base, 2010, ADA516706, http://handle.dtic.mil/100.2/ADA516706. Develops and measures a packet reception kernel module and with logging to the file system.
3. [Mrazek08] Tomas Mrazek and Jan Vykopal, “Packet Capture Benchmark on 1 GE,” CESNET Technical Report 22/2008, http://www.cesnet.cz/doc/techzpravy/2008/packet-capture-benchmark/packet-capture-benchmark.pdf. Detailed report on performance of of packet reception using the

COMBO6 card.

4. [Gasparakis10] Joseph Gasparakis and Peter P Waskiewicz, Jr., “Design considerations for efficient network applications with Intel® multi-core processor-based systems on Linux,” Intel Embedded Design Center White Paper, 324176-001US, http://edc.intel.com/Link.aspx?id=3680. Looks at what can effect the performance of Intel 1 GbE and 10 GbE network card and suggests kernel configuration options to improve performance.
5. [Jain91] R.K. Jain, The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling, Wiley, 1991. Defines how systems should be analyzed and what pieces of the system to look at.
6. [Deri04] Luca Deri, Netikos S. P. A, Via Del Brennero Km, and Loc La Figuretta, “Improving Passive Packet Capture: Beyond Device Polling,” Proceedings of SANE 2004, http://luca.ntop.org/Ring.pdf. Develops the `PF_RING` packet capture method that uses a buffer between user- and kernel-space for more efficient utilization.
7. [Fusco10] Francesco Fusco and Luca Deri, “High Speed Network Traffic Analysis with Commodity Multi-core Systems,” Internet Measurement Conference 2010, http://conferences.sigcomm.org/imc/2010/papers/p218.pdf. Takes advantage of the multiple receive queues on modern network cards and multi-core systems to increase throughput.
8. [Schneider07] Fabian Schneider, JÃ¶rg Wallerich and Anja Feldmann, “Packet Capture in 10-Gigabit Ethernet Environments Using Contemporary Commodity Hardware,” Lecture Notes in Computer Science, 2007, Volume 4427/2007, 207-217, http://dx.doi.org/10.1007/978-3-540-71617-4_21. Uses flow distribution on the switch to “fairly” partition 10 Gbps link among 10-1 Gbps systems.
9. [Huang10] Chungang Huang, Xiangzhan Yu, and Hao Luo, “Research on high-speed network data stream capture based on multi-queue NIC and multi-core processor,” The 2nd IEEE International Conference on Information Management and Engineering (ICIME), 2010, http://dx.doi.org/10.1109/ICIME.2010.5477440. Develops a special purpose operating system to take complete control of the processor to achieve 1 Gbps throughput.
10. [Olsson05] Robert Olsson, “pktgen the linux packet generator,” Ottawa Linux Symposium 2005, http://www.kernel.org/doc/ols/2005/ols2005v2-pages-19-32.pdf. Explains the Linux kernel packet generator that can be used to generate packets at near line-speed.
11. [Jones08] M. Tim Jones, “Anatomy of Linux loadable kernel modules: A 2.6 kernel perspective,” IBM developerWorks Linux Technical Library, July 16, 2008, http://www.ibm.com/developerworks/linux/library/l-lkm/index.html. Explains how Linux kernel modules are loaded into the running kernel.
12. [Mogul97] Jeffrey C. Mogul and K.K. Ramakrishnan, “Eliminating receive livelock in an interrupt-driven kernel,” ACM Transactions on Computer Systems, Volume 15 Issue 3, August 1997, http://dx.doi.org/10.1145/263326.263335. Observes that, under heavy packet load, a system can be put in a state where it cannot make forward progress leading to dropped packets.
13. [Salim01] Jamal Hadi Salim, Robert Olsson, and Alexey Kuznetsov, “Beyond Softnet,” USENIX Proceeding of the 5th Annual Linux Showcase & Conference, http://www.usenix.org/publications/library/proceedings/als01/full_papers/jamal/jamal.pdf. Solves the livelock problem in Linux by using adaptive polling through the New Applications Programming Interface.
14. [Intel06] Intel, “Accelerating High-Speed Networking with Intel® I/O Acceleration Technology,” White Paper, 316125-001US, 2006, http://download.intel.com/support/network/sb/98856.pdf. Abstract view of Intel's IO Acceleration Technology.
15. [Sysstat] “SYSSTAT,” http://sebastien.godard.pagesperso-orange.fr/, accessed April 11, 2011. Linux command line tool kit that periodically determines the per-core processor load.
16. [Endace] “EndaceProbes – Purpose built for high-speed packet capture,” http://www.endace.com/high-speed-packet-capture-hardware.html. Developer of hardware-based packet capturing method.
17. [Napatech] “40 GbE Packet Capture & Replay Adapter,” http://www.napatech.com/products/capture_adapters/1x40g_pcie_nt40e2-1_capture.html. Another developer of hardware-based packet capturing.
18. [WildPackets] “OmniAdapter 1G Network Analyzer,” http://www.wildpackets.com/products

/network_recorders/omniadapter_analysis_cards/omniadapters. A third developer of hardware-based packet capturing.
19. [ONL] Open Network Laboratory, http://onl.wustl.edu/, 2011. Guide to using Washington University's Open Network Laboratory to perform network-based performance measurements.

List of Acronyms

ANOVA	Analysis of Variance
CPU	Central Processing Unit
DDR	Double data rate
ECC	Error-correcting code
GbE	Gigabit (per second) Ethernet
Gbps	Gigabit per second
GHz	Gigahertz
GiB	Gibibyte (2^{30} Bytes)
IP	Internet Protocol
(soft)IRQ	(software) Interrupt Request
KiB	Kibibyte (2^{10} Bytes)
LKPG	Linux Kernel Packet Generator
MHz	Megahertz
MiB	Mebibyte (2^{20} Bytes)
NAPI	New Application Programming Interface
NIC	Network Interface Card
ONL	Open Network Laboratory
O/S	Operating System
PCAP	Packet Capture
PCI	Peripheral Component Interconnect
SUT	System Under Test
TCP	Transmission Control Protocol

Last modified on April 24, 2011
This and other papers on latest advances in performance analysis are available on line at
http://www1.cse.wustl.edu/~jain/cse567-11/index.html
Back to Raj Jain's Home Page