Packet Capture in 10-Gigabit Ethernet Environments Using Contemporary Commodity Hardware

Fabian Schneider, Jörg Wallerich, and Anja Feldmann

Deutsche Telekom Laboratories / Technische Universität Berlin
10587 Berlin, Germany
{fabian,joerg,anja}@net.t-labs.tu-berlin.de
http://www.net.t-labs.tu-berlin.de

Abstract. Tracing traffic using commodity hardware in contemporary high-speed access or aggregation networks such as 10-Gigabit Ethernet is an increasingly common yet challenging task. In this paper we investigate if today's commodity hardware and software is in principle able to capture traffic from a fully loaded Ethernet. We find that this is only possible for data rates up to 1 Gigabit/s without reverting to using special hardware due to, e. g., limitations with the current PC buses. Therefore, we propose a novel way for monitoring higher speed interfaces (e. g., 10-Gigabit) by distributing their traffic across a set of lower speed interfaces (e. g., 1-Gigabit).

This opens the next question: which system configuration is capable of monitoring one such 1-Gigabit/s interface? To answer this question we present a methodology for evaluating the performance impact of different system components including different CPU architectures and different operating system. Our results indicate that the combination of AMD Opteron with FreeBSD outperforms all others, independently of running in single- or multi-processor mode. Moreover, the impact of packet filtering, running multiple capturing applications, adding per packet analysis load, saving the captured packets to disk, and using 64-bit OSes is investigated.

Keywords: Packet Capturing, Measurement, Performance, Operating Systems.

1 Introduction

A crucial component of almost any network measurement and especially any network security system is the one that captures the network traffic. For example, nowadays almost all organizations secure their incoming/outgoing Internet connections with a security system. As the speed of these Internet connection increases from T3 to 100-Megabit to 1-Gigabit to 10-Gigabit Ethernet, the demands on the monitoring system increase as well while the price pressure remains. For example, the Münchner Wissenschaftsnetz (MWN, Munich Scientific Network) [1] offers Internet connection to roughly 50,000 hosts via a 10-Gigabit bidirectional uplink to the German Scientific Network (DFN). While the link is currently rate limited to 1.2-Gigabit we indeed face the challenge of performing packet capture in a 10-Gigabit Ethernet environment using commodity hardware in order to run a security system. We note that most network security systems rely on capturing *all* packets as any lost packet may have been the attack. Furthermore, most attack detection mechanisms rely on analyzing the packet content and the security system itself has to be resilient against attacks [2]. Therefore packet capture has to be able to handle both the maximum data rates as well as the maximum packet rates.

Unfortunately, network traffic capture is not an easy task due to the inherent system limitations (memory and system bus throughput) of current off-the-shelf hardware. One expensive alternative[1] is to use specialized hardware, e. g., the monitoring cards made by Endace [3]. Our experience shows that PCs equipped with such cards are able to capture full packet traces to disk in 1-Gigabit environments. But even these systems reach their limits in 10-Gigabit environments: they lack CPU cycles to perform the desired analysis and/or bus bandwidth to transfer the data to disk. Therefore, we in this paper propose a novel way for distributing traffic of a high-speed interface, e. g., 10-Gigabit, across multiple lower speed interfaces, e. g., 1-Gigabit, using of-the-shelf Ethernet switches. This enables us to use multiple machines to overcome the system limitations of each individual interface without losing traffic.

This leaves us with the question, which system is able to monitor one such 1-Gigabit Ethernet interface given the many factors that impact the performance of a packet capture system. These include the system, CPU, disk and interrupt handling architecture [4], the operating system and its parameters, the architecture of the packet capture library [5,6,7,8], and the application processing the data [2,5,9]. In this paper, we focus on the first set of factors and only consider simple but typical application level processing, such as compressing the data and storing it to disk. We chose to examine three high-end off-the-shelf hardware architectures (all dual processor): Intel Xeon, AMD Opteron single core and AMD Opteron multi-core based systems; two operating systems: FreeBSD and Linux with different parameter settings; and the packet capture library, libpcap [5], with different optimizations.

In 1997, Mogul et al. [4] pointed out the problem of receive livelock. Since then quite a number of approach [10,11,7,8] to circumvent this problem have been proposed. Yet the capabilities and limitations of the current capturing systems have not been examined in the recent past.

The remainder of this paper is structured as follows. In Sec. 2 we discuss why capturing traffic from an 10-Gigabit Ethernet link is nontrivial and propose ways of distributing traffic across several 1-Gigabit Ethernet links. In Sec. 3 we ask if it is feasible to capture traffic across 1-Gigabit links using commodity hardware. Next, in Sec. 4, we discuss our measurement methodology and setup. Sec. 5 presents the results of our evaluation. Finally, we summarize our contributions and discuss future work in Sec. 6.

2 Challenges That Hinder Data Capture at High Rates

With current PC architectures there are two fundamental bottlenecks: bus bandwidth and disk throughput. In order to capture the data of a fully utilized bidirectional 10-Gigabit link one would need 2560 Mbytes/s system throughput. Even, for capturing only the packet headers, e. g., the first 68 bytes, one needs 270 Mbytes/s given an observed average packet size of 645 bytes. Moreover, when writing to disk, packet capture

[1] Current cost for a 1-Gigabit card is roughly 5,000 €.

libraries require the data to pass the system bus twice: Once to copy the data from the card to the memory to make it accessible to the CPU, and once from memory to disk. Thus this requires twice the bandwidth.

When comparing these bandwidth needs with the bandwidth that current busses offer we notice a huge gap, especially for full packet capture. The standard 32-bit, 66 MHz PCI bus offers 264 Mbytes/s. The 64-bit, 133 MHz PCI-X bus offers 1,066 Mbytes/s. The PCIexpress x1 bus offers 250 Mbytes/s. It is easy to see that none of these busses are able to handle the load imposed by even a single uni-directional 10-Gigabit link (full packets). Furthermore the numbers have to be taken with a grain of salt as they are maximum transfer rates, which in case of PCI and PCI-X are shared between the attached PCI cards. Some relief is in sight, PCIexpress x16 (or x32) busses which offer 4000 Mbytes/s (8000 Mbytes/s) are gaining importance. Currently boards with one or two x16 slot are available. Yet, note that these busses are intended for graphic cards and that 10-Gigabit Ethernet cards are only available for the PCI-X bus at the moment. Indeed, at this point we do not know of any network or disk controller cards that support PCIexpress x16 or x32.

But what about the bandwidth of the disk systems. The fastest ATA/ATAPI interfaces runs at a mere 133 Mbytes/s; S-ATA (2) offers throughputs of up to 300 Mbytes/s; SCSI (320) can achieve 320 Mbytes/s. Even the throughput of Fiberchannel, up to 512 Mbytes/s, and Serial-Attached-SCSI (SAS), up to 384 Mbytes/s, is not sufficient for 10-Gigabit links. The upcoming SAS 2 systems are going to offer 768 Mbytes/s while SAS 3 systems may eventually offer 1536 Mbytes/s. Again there is a huge gap.

Therefore, it appears unavoidable to distribute the load across multiple machines. Instead of designing custom hardware for this purpose we propose to use a feature of current Ethernet switches. Most switches are capable of bundling a number of lower speed interfaces. Once the bundle has been configured it can be used like a normal interface. Therefore it can be used as a monitor interface for any other interface. Accordingly, the switch forwards all traffic that it receives on a, e. g., 10-Gigabit interface on a bundle of, e. g., 1-Gigabit interfaces. These lower speed interfaces can then be monitored individually. Fig. 1 shows a schematic view of this setup. There are three major drawbacks to this solution: first, time stamps are slightly distorted as the packets have to pass through an additional switch; second, it is hard to synchronize the system time of the monitors; third, even though the overall capacity suffices the individual capacity of each link may not be sufficient. This depends on the load balancing scheme in use.

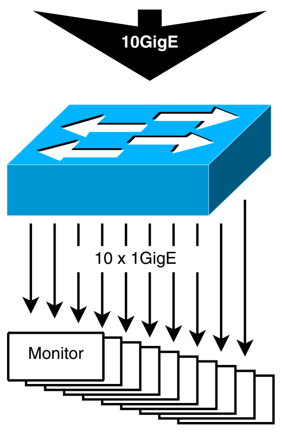

Fig. 1. Distr. Setup

We tested this mechanism by using a Cisco 3750 switch to move from monitoring a single 1-Gigabit Ethernet link to eight (which is maximum for this vendor) 100-Mbit links. We use the EtherChannel feature and configured such a link bundle across eight 100-Mbit links. Then this link is used as a monitor link for a 1-Gigabit input link.

It is crucial to use the appropriate load balancing method for this link. Common options include load balancing based on MAC addresses (simple switches), IP addresses and/or MAC addresses (e. g., Cisco 3750), combinations of IP addresses and/or port (TCP/UDP) numbers etc. (Cisco 6500 Series). Note that per packet multiplexing is not a sensible option as this can lead to an uneven distribution across the monitors. Given that it is common to use a switch for monitoring traffic traveling between two routers the MAC address variability is limited. There are usually only two MAC addresses in use, one for each endpoint. Accordingly, we have to rely on at least source and destination IP addresses or better yet IP addresses and port numbers for load balancing which unfortunately rules out the cheapest switches. Still, depending on the application one may want to ensure that all packets from an IP address pair or a remote or local host are handled by a single monitor. This ensures that all packets of a single TCP connection arrive at the same monitor. With such a configuration load balancing can be expected to be reasonable as long as there are sufficiently many addresses that are monitored. If the EtherChannel feature is not supported by a specific switch one should check if it offers another way to bundle links. A good indication that a switch is capable of bundling is the support of the Link Aggregation Control Protocol (LACP).

3 Influencing Parameters and Variables

Given that we now understand how we can distribute the traffic from an 10-Gigbit interfaces across multiple 1-Gigabit interfaces we next turn our attention to identify suitable commodity hard- and software for the task of packet capturing. For this purpose we identify the principal factors that may impact the performance and then examine the most critical ones in detail.

Obvious factors include the network card, the memory speed, and the CPU cycle rate. A slightly less obvious one is the system architecture: does it matter if the system has multiple CPUs, multiple cores, or if hyper-threading is enabled? Another important question is how the OS interacts with the network card and passes the captured packets from the card to the application. How much data has to be copied in which fashion? How many interrupts are generated and how are they handled? How much buffer is available to avoid packet loss? Yet another question is if a 64-bit OS offers an advantage over the 32-bit versions. With regards to the application we ask by how much the performance is degraded when running multiple capturing applications on the same network card, when adding packet filters, when touching the data in user space, i. e., via a copy operation or by compressing them, and when saving the data to a tracefile on disk. These are the questions that we try to answer in the remainder of the paper.

4 Setup

Conducting such measurements may appear simple at first glance, but the major difficulty is to find hard-/software configurations that are comparable. After all we want to avoid comparing apples to oranges.

4.1 Hardware and Software

To enable a fair comparison, state-of-the-art off-the-shelf computer hardware was purchased at the same time, February 2004, with comparable components. Preliminary experiments have shown that the Intel Gigabit Ethernet cards provide superior results than, e. g., those of Netgear. Furthermore, 2 Gbytes of RAM suffice and we choose to use the fastest available ones. Accordingly, we focus on the system architecture and the OS while using the same system boards, the same amount of memory, and the disk system. Consequently, we purchased two AMD Opteron and two Intel Xeon systems[2] that are equipped with the same optical network card, the same amount of memory, and the same disk system (IDE ATA based). One Opteron and one Xeon system were installed with FreeBSD 5.4 and the others were installed with Debian Linux (Kernel v2.6.11.x).

Once dual-core systems became available we got two additional machines in May 2006: HP Proliant DL385[3]. In contrast to the other systems these have an internal SCSI Controller with three SCSI disk attached. One was installed with FreeBSD 6.1 and the other with Debian Linux (Kernel v2.6.16.16), both with dual-boot for 32-bit and 64-bit.

4.2 Measurement Topology

To be able to test multiple systems at the same time and under exactly the same workload we choose the setup shown in Fig. 2. A workload generator, see Sec. 4.4, generates traffic which is fed into a passive optical splitter. The task of the splitter is to duplicate the traffic such that all systems under test (SUT) receive the same input. The switch between the workload generator and the splitter is used to check the number of generated packets while all capture applications running on the SUT's are responsible for reporting their respective capture rate. In later experiments we removed the switch as the statistics reported by the traffic generator itself proved to be sufficient. The control network is a separate network and is used to start/end the experiments and to collect the statistics.

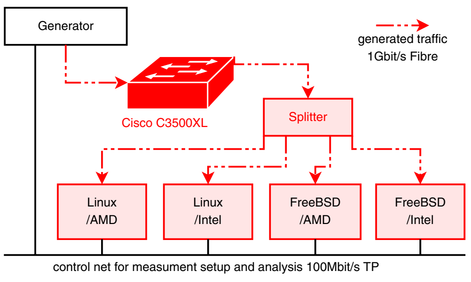

Fig. 2. Measurement Topology

4.3 Methodology

Each measurement varies the traffic rate from 50 Mbit/s to 920 Mbit/s[4] and consists of seven repetitions of sending one million packets at each bandwidth setting to reduce

[2] System details: AMD Opteron 244, Intel Xeon 3.06Ghz, 2 Gbytes RAM (DDR-1 333 MHz), Intel 82544EI Gigabit (Fiber) Ethernet Controller, 3ware Inc. 7000 series ATA-100 Storage RAID-Controller with at least 450 Gbytes space.

[3] Details see previous footnote except of processors (AMD Opteron 277), disk system (HP Smart Array 64xx Controller with 3×300 Gbytes U320 SCSI hard disks), and faster memory.

[4] 920Mbit/s is close to the maximum achievable theoretical input load given the per packet header overheads.

statistical variations. In each iteration we start the capture process (using a simple libpcap [5] based application [12] or tcpdump) and a CPU usage profiling application on the systems under study. Then we start the traffic generator. Once all packets of the run have been sent by the workload generator we stop the packet capture processes.

For each run we determine the capture rate, by counting the received number of packets, and the average CPU usage (see `cpusage` [12]). The results are then used to determine the mean capturing rate as well as the mean CPU usage for each data rate.

4.4 Workload

With regards to the generated traffic we decided to neither focus on the simplest case for packet monitoring (all large packets) nor on the most difficult one (all small packets) as both are unrealistic. The first case puts too much weight on the system bus performance while the latter one emphasizes the interrupt load too much.

Instead we decided to use a traffic generator that is able to generate a packet size distribution that closely resembles those observed in actual high-speed access networks. For this purpose we enhanced the Linux Kernel Packet Generator (LKPG [13]), which is capable of filling a 1-Gigabit link, to generate packets according to a given packet size distribution.

Figure 3 shows the input and the generated packet size distributions of our modified LKPG based on a 24h packet level trace captured at the 1-Gigabit uplink of the MWN [1] (peaks are at 40–64, 576 and 1420–1500 B).

We decided to not mimic flow size distribution, application layer protocol mix, delay and jitter, or data content, as they have no direct impact on the performance of the capture system. Their impact is on the application which is beyond the scope of this paper. The same holds for throughput bursts, which can be buffered. The intention of the work is to identify the maximum throughput a capturing system can handle. We realize different traffic rates by inserting appropriate inter-packet gaps between the generated packets.

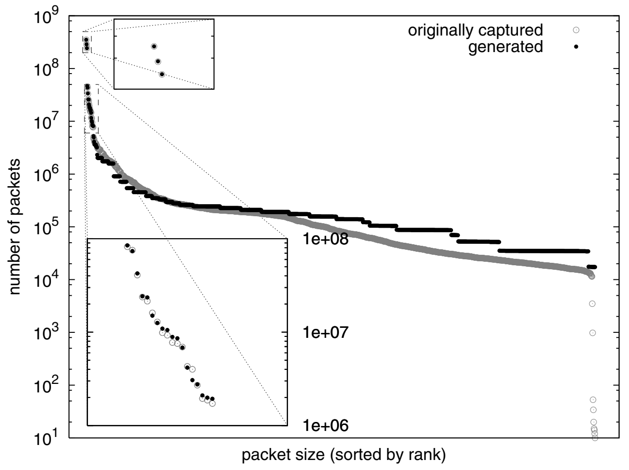

Fig. 3. Packet sizes (sorted by rank) vs. frequency of the packets (y-axis in logscale)

5 Results

We now use the above setup to evaluate the performance of our systems starting from the vanilla kernels. We find that it is crucial to increase the amount of memory that is available to the kernel capture engine. Otherwise scheduling delays, e. g., to the capturing application, can cause the overflow of any of the kernel queues, e. g., the network queue for the capture socket. This is achieved by either setting the `debug.bpf_bufsize` sysctl parameter (FreeBSD) or the `/proc/sys/net/core/rmem*` parameter (Linux). Based upon our experience we chose to use buffers of 20 Mbytes. Furthermore, we noticed that the systems spend most of their cycles on interrupt processing when capturing at high packet rates. To overcome this problem we tried device polling as suggested by Mogul et al. [4]. For Linux this reduces the CPU cycles that are spent in kernel mode and thus increases the packet capture rate significantly. For FreeBSD activating the polling mechanism slightly reduced the capturing performance and the stability of the system. Therefore we use it for Linux but not for FreeBSD.

Next we baseline the impact of the system architecture and the OS by comparing the performance of the four systems with single processor kernels. Figure 4 (top) shows the capture rate while Fig. 4 (bottom) shows the CPU usage as we increase the data rate. To keep the plots simple we chose to not include the standard deviation. Note that all measurements have a standard deviation of less than 2%. As expected the capture rate decreases while the CPU usage increases as the data rate is increased. Unfortunately, only the FreeBSD/AMD combination loses almost no packets. All others experience significant packet drops. We note that the systems start dropping packets once their CPU utilization reaches its limits. This indicates that the drops are not due to missing kernel buffers but are indeed due to missing CPU cycles. The FreeBSD/Intel combination already starts dropping packets at about 500 Mbit/s. Neither of the Linux systems looses packets until roughly 650 Mbit/s. From that point onward their performance deteriorates

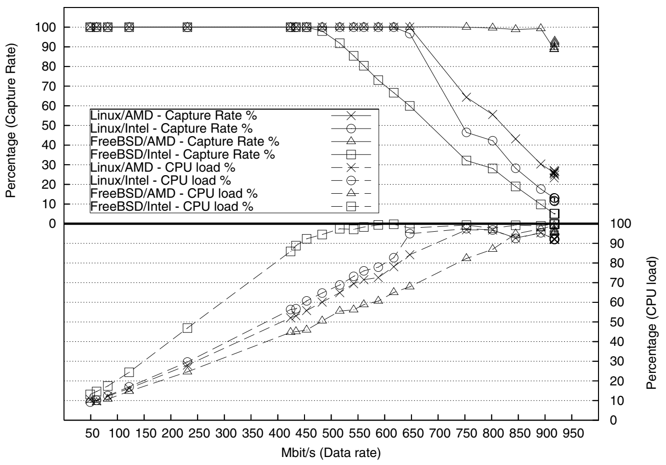

Fig. 4. Data rate vs. Capture Rate (top) and CPU utilization (bottom) for: single processor; increased buffers

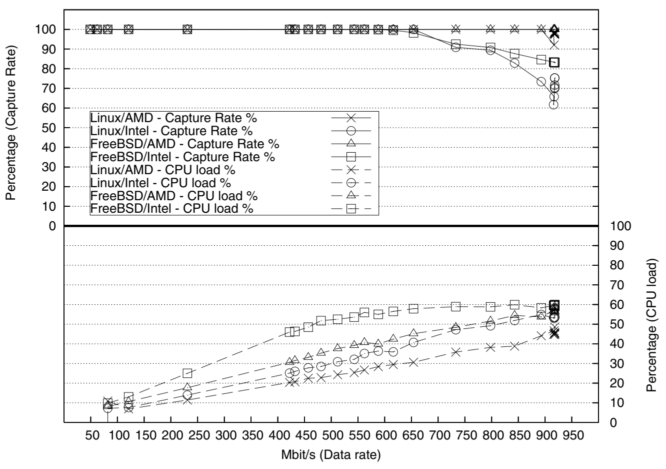

Fig. 5. Data rate vs. Capture Rate (top) and CPU utilization (bottom) for: multiple processors; increased buffers. (A CPU usage of 50% implies that one CPU is fully utilized.)

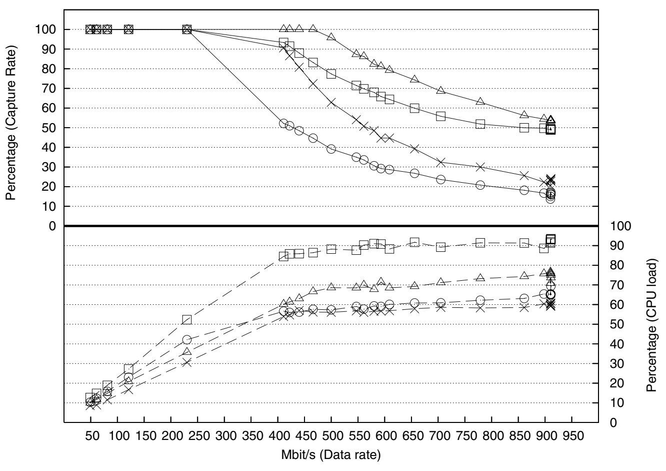

Fig. 6. Data rate vs. Capture Rate (top) and CPU utilization (bottom) for: multiple processors; increased buffers; 50 additional `memcpy` operations on the packet data. (To increase readability the legend for this plot is omitted as it is the same as in Fig. 4.)

dramatically with increasing data rates. The efficiency of the FreeBSD/AMD machine is especially surprising as FreeBSD performs an additional (kernel) packet copy operation and does not use device polling which proved beneficial on Linux systems.

This observation indicates that utilizing the multi-processor system may help overcome the above problems as the kernel can be scheduled on one processor while the application is scheduled on the other. This almost doubles the CPU resources. Yet,

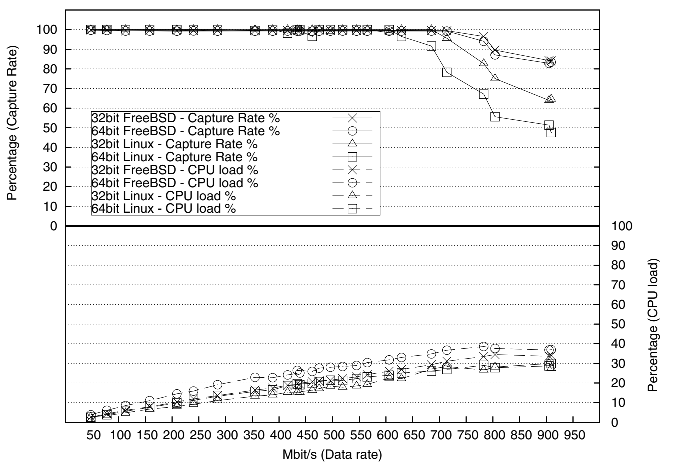

Fig. 7. Data rate vs. Capture Rate (top) and CPU utilization (bottom) for: multiple *dual-core* processors; increased buffers; writing full packets to disk. (A CPU usage of 25% implies that one CPU is fully utilized.)

memory conflicts and cache misses have the potential to deteriorate the capture performance. The results shown in Fig. 5 show the resulting significant performance increase. This indicates that the additional CPU cycles of the SMP architecture clearly top their penalties, e. g., cache misses. In fact, the FreeBSD/AMD combination is able to capture all packets even at the highest data rate. Overall we note that in our experience FreeBSD outperforms Linux. To test if further increasing the multiprocessor capabilities helps the performance we enabled Hyper-Threading (only available on Intel machines). This does not impact the performance. But keep in mind that in this test only two processes require significant CPU resources, the kernel and the capture application.

As it is common to use filters while capturing, we studied the impact of configuring a 50 BFP instructions filter. We find that even if packets have to pass a long filter before being accepted that does not drastically change the performance. The performance of the FreeBSD machines stays as is while the one of the Linux machines decreases slightly (up to 10%) at high data rates (>800 Mbits/s). This indicates that at least for FreeBSD filtering does not impose high costs on CPU resources. But what if we run multiple capture applications at the same time? In this case the packets have to be duplicated within the kernel and delivered to multiple filter instances. We not surprisingly find that the performance deteriorates for all four systems. As the Linux machines start to massively drop packets when reaching their CPU limits, we suggest to use FreeBSD as the drop rates are less significant.

The next challenge that we add to the application are memory copy operations as these are common in network security applications. To investigate an extreme position we instructed the application to copy the captured packet 50 times in user-space. From Fig. 6 we see that all systems suffer under the additional load. The Linux machines yet again experience larger problems. The next challenge is to compress the content of each packet using a standard compression routine, `libz`. For the first time the Intel systems outperform their AMD counterparts. This suggests that Intel processors and not AMD's have better hardware realizations for instructions used by `libz`.

Recapitulating, we find that the multiprocessor Opteron system with FreeBSD 5.4 outperforms all other systems. Its maximum loss rate is less than 0.5%. One explanation is that AMD has a superior memory management and bus contention handling mechanism. These results motivated us to only purchase Opteron dual-core machines for the next tests. To baseline the new system we repeated the above experiments, except the ones with additional application load. These experiments show that both machines can capture every single packet on FreeBSD as well as Linux running either the 32-bit or the 64-bit version of the respective OS.

The newer machines clearly outperformed the older ones. Therefore we next examine if either system is able to capture full packet traces to disk. For this we switch to using `tcpdump` as application. From Fig. 7 we see that Linux is able to write all packets to disk up to a data rate of about 600 Mbit/s independent of using a 32-bit or 64-bit kernel. FreeBSD is always dropping roughly 0.1% of the packets even at the lowest data rates. This indicates a minor but fixable principle problem in the OS. FreeBSD only begins to drop a significant number of packets when the data rate exceeds 800 Mbit/s. But keep in mind that Linux captures only about 65% of the packets at the highest data rate (32-bit). While under FreeBSD the difference between 32-bit and 64-bit mode is negligible (up to a capture rate of 83%), Linux in 64-bit mode deteriorates drastically. It records only half of the generated packets. The poor performance of 64-bit Linux might be due to the increased memory consumption for longer pointers within the kernel.

6 Summary

In this paper we argue that due to currently available bus and disk bandwidth it is impossible to tackle the crucial problem of packet capture in a 10-Gigabit Ethernet environment using a single commodity system. Therefore we propose a novel way for distributing traffic across a set of lower speed interface using a switch feature that allows one to bundle lower speed interface into a single higher speed interface, e. g., Cisco's EtherChannels feature. Each of the lower speed interfaces can then be monitored using commodity hardware.

To answer the question which system is able to support packet monitoring best we present a methodology for evaluating the performance impact of various system components. We find that AMD Opteron systems outperform Intel Xeon ones and that FreeBSD outperforms Linux. While multi-processor systems offer a lot, the benefit of adding Hyper-Threading and multi-core is small. Moreover, it appears that for the task of packet capture the 64-bit OS versions are not quite ready yet. The newer systems clearly outperform the older ones and can even capture full packet traces to disk as long as the data rate is less than 600 to 700 Mbit/s. All in all, multi-processor Opteron systems with FreeBSD clearly outperformed all other systems.

Obviously, our goal has to be to understand not just the packet capture performance but the characteristics of such highly complex security screening applications as Bro.

Therefore we plan to investigate how system performance scales with the traffic load and if there is a way to predict future performance. A summary of our current and future results is available on the project Website [14].

References

1. The Munich Scientific Network. `http://www.lrz-muenchen.de/wir/intro/en/#mwn`
2. Paxson, V.: Bro: A System for Detecting Network Intruders in Real-Time. Computer Networks, **31(23-24)** (1999) 2435–2463
3. Endace Measurement systems: `http://www.endace.com`
4. Mogul, J. C., and Ramakrishnan, K. K.: Eliminating receive livelock in an interrupt-driven kernel. ACM Transactions on Computer Systems, **15(3)** (1997) 217–252.
5. Jacobson, V., Leres, C., and McCanne, S.: libpcap and tcpdump. `http://www.tcpdump.org`
6. Wood, P.: libpcap MMAP mode on linux. `http://public.lanl.gov/cpw/`
7. Deri, L.: Improving passive packet capture: Beyond device polling. In *Proc. of the 4th Int. System Administration and Network Engineering Conference (SANE'2004)* (2004)
8. Deri, L.: nCap: Wire-speed packet capture and transmission. In *Proc. of the IEEE/IFIP Workshop on End-to-End Monitoring Techniques and Services (IM 2005, E2EMON)* (2005)
9. Snort `http://www.snort.org/`
10. Salim, H. D., Olsson, R., and Kuznetsov, A.: Beyond softnet. In *Proc. of the 5th Annual Linux Showcase & Conference* (2001)
11. Rizzo, L.: Device Polling support for FreeBSD. In *Proc. of the EuroBSDCon' 01* (2001)
12. Schneider, F.: Performance Evaluation of Packet Capturing Systems for High-Speed Networks Diploma thesis, Technische Universität München (2005) for `cpusage` and the capturing application see: `http://www.net.in.tum.de/~schneifa/proj_en.html`
13. Olsson, R.: Linux kernel packet generator.
14. Hints for improving Packet Capture System performance: `http://www.net.t-labs.tu-berlin.de/research/bpcs/`