SAND2011-5589C

Experimental Evaluation of the Impact of Packet Capturing Tools for Web Services

Chao-Chih Chen[◇] [†], Yung Ryn Choe [*], Chen-Nee Chuah[‡] and Prasant Mohapatra[†]
[†]Department of Computer Science. University of California, Davis, CA 95616
Email {cchchen,pmohapatra}@ucdavis.edu
[*]Sandia National Laboratories. Livermore, CA 94551
Email yrchoe@sandia.gov
[‡]Department of Electrical Engineering. University of California, Davis, CA 95616
Email: chuah@ucdavis.edu

***Abstract*—Network measurement is a discipline that provides the techniques to collect data that are fundamental to many branches of computer science. While many capturing tools and comparisons have made available in the literature and elsewhere, the impact of these packet capturing tools on existing processes have not been thoroughly studied. While not a concern for collection methods in which dedicated servers are used, many usage scenarios of packet capturing now requires the packet capturing tool to run concurrently with operational processes.**

In this paper we perform experimental evaluations of the performance impact that packet capturing process have on web-based services; in particular, we observe the impact on web servers. We find that packet capturing processes indeed impact the performance of web servers, but on a multi-core system the impact varies depending on whether the packet capturing and web hosting processes are co-located or not. In addition, the architecture and behavior of the web server and process scheduling is coupled with the behavior of the packet capturing process, which in turn also affect the web server's performance.

I. Introduction

Network measurement is a discipline that provides the foundation for many studies in networked systems. From capacity planning to anomaly detection to network security, being able to measure and collect data from the network is crucial for the success in these tasks. One such popular method to collect data from the network is packet capturing. Because the collected data (the packet) contains application-invariant and application-specific information, it is a good candidate for one-time data collection that can support various types of analysis. In addition, packet capturing tools are widely available (e.g., Wireshark, TCPDump for Linux, NetMon for Windows), and there are mature libraries for custom codes to tap into the packet monitoring process.

In the past network measurements are often collected by means of port mirroring at the router and dedicated machines to collect the packets. Another way to monitor the network is to do so at the edge of the network (i.e., capture at the server machines). This reduces the need to have a dedicated machines and also amortizes the cost of packet capturing over all the machines.

Although not a concern if dedicated machines are used to capture packets, the performance impact of capturing process becomes important if it is to co-locate in the same physical machine as running processes. In fact, in some cases monitoring at the server machines is preferred, if not essential. For example, there are works that attempt to discover application-level dependency [1], [2] while others try to localize the source of faults from information derived from captured packets [3]. Without capturing the network information at the server machines, details such as application-level dependency is either impossible or much more difficult to capture elsewhere.

Recognizing the deficiency of research work in this area, we carry out experiments to examine the impact that packet capturing process has on web-based services. In particular, we test packet capturing process' performance impact on web servers at their saturation point. This gives us some insight into the maximum performance achievable for web services when packet capturing process is also running, and whether it adversely impacts existing services.

The contributions of this paper are:

- Experimental evaluation on the performance impact of capturing process to co-located web-based services.
- Deployment of two web servers of different architecture to validate that results are consistent across different web server architectures.
- Measurement of both system-level statistics and user-perceived statistics to observe correlation between the two.

In Section II we briefly go over the packet capturing process; Section III presents the evaluation methodology and the results obtained; Section IV discuss the related works; Section V discusses future works and concludes the paper.

II. Packet Capturing Process

This section provides a high-level overview of the packet capturing process in Linux, as to make the discussion in this paper complete. The goal of this section is not to provide a complete detail of the internals of packet capturing, but to illuminate on the steps involved in delivering the packet from

[◇] Is supported in part by fellowship from Sandia National Laboratories.

*Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energys National Nuclear Security Administration under Contract DE-AC04-94AL85000.

the network card to the kernel and to the user application. This will help towards understanding the behaviors observed in Section III; details of the packet capturing process described here can be found in [4], [5].

When packets are transmitted over the wire, the network interface card (NIC) normally picks up the packet if the packet is destined for it; under promiscuous mode it will pick up all packets sensed. Once the packet is recognized for reception, the NIC's interrupt routine is invoked, in which the routine allocates some space in memory and copies the packet into the allocated memory. The packet is not immediately processed after moving to the memory, as the interrupt routine is intended to perform as little operation as possible. When the packet is picked up later by the software interrupt handler, it passes the packet upwards to the appropriate protocol handler based on the packet's protocol type.

For packets destined for packet capturing tools, a special handler is used so that all the packets can be handled and subsequently forwarded to the capturing process. Corresponding to this special handler is a special protocol family called PF_PACKET, and the packet is copied[1] and delivered to a socket created specifying the PF_PACKET family. Copying is needed because the packet might be actually destined for an application at the local machine, so the packet must be copied for separate consumption by the capturing process and the application.

While this overview is brief, it illuminates the many transactions involved in capturing the packet, and these transactions will result in the use of CPU resource. The experiments to be discussed in Section III are aimed at observing how its uptake of CPU resource affects the performance of web servers.

III. Evaluation

In this section we present the experimental results obtained when running the packet capturing tool under various application scenarios. We will first discuss the set-up of our experiments and the metrics we set out to collect; then we will discuss the results of the experiments and various inferences drawn from them.

A. Experimental Set-up

To observe the effects of packet capturing on applications, we deployed two web servers and collected various performance metrics. The web servers used are Apache version 2.2.17 with worker MPM [6] and Nginx version 0.8.54 [7]; the kernel used is 2.6.35.11. We choose to use Apache because it is rated the most-used web server according to the latest survey by NetCraft (February 2011 at the time of writing) [8]; while Nginx is also used because its architecture is fundamentally different from Apache.

The Apache architecture offers various Multi-Processing Modules (MPM) as a way to scale the web server with increasing user demand. These MPMs are either multi-processed, multi-threaded, or both. In the model we use, the worker module[2] is selected. Apache uses a parent process to accept incoming connections and distributes them to multiple processes/threads, where each process/threads handles one request at a time [9][3]. One implication of such an architecture means that in order to scale, the number of threads and process needed to spawn needs to increase. On the other hand, Nginx [7] is a highly-scalable web server developed to address the ability to serve high number of simultaneous user connections, also known as the C10K problem [10] Nginx operates under the asynchronous call model, so a single process can scale quite well against increasing concurrent request volume.

To measure the performance impact of packet capturing, we monitor the CPU and bandwidth utilization of all applications, and application-specific metrics. The packet capturing tool we used in this experiment is a vanilla TCPDump, as we would like to observe the impact of a free and widely available tool without any performance modification to it (e.g., mmap extension, PF_RING extension [11]). To stress the web servers, we use HTTP_LOAD [12] to constantly fetch a small static HTML page from the servers. In addition, for all the experiments the web server is hosted on a Duo Core system, but we confine the web server to a single core. This allows us to experiment with co-locating and separating the packet capturing and web server process. By doing so, we can observe the effect that sharing CPU resource has on the performance degradation of the web servers.

For each web server, we test the co-location factor by moving TCPDump and the web server processes onto the same or different CPU core, and we also vary TCPDump's behavior in four ways: no TCPDump running, normal TCPDump, quieter TCPDump, and TCPDump writing captured packets to disk. We have a total of eight settings, two of which are repeats (no TCPDump and co-locate is the same as no TCPDump and do not co-locate). For each setting we use HTTP_LOAD on two different clients, where each client set the HTTP_LOAD concurrency to 15, to fetch continuously from the server for five minutes; metrics are collected every five seconds to minimize its impact on the experiment.

B. Web Servers

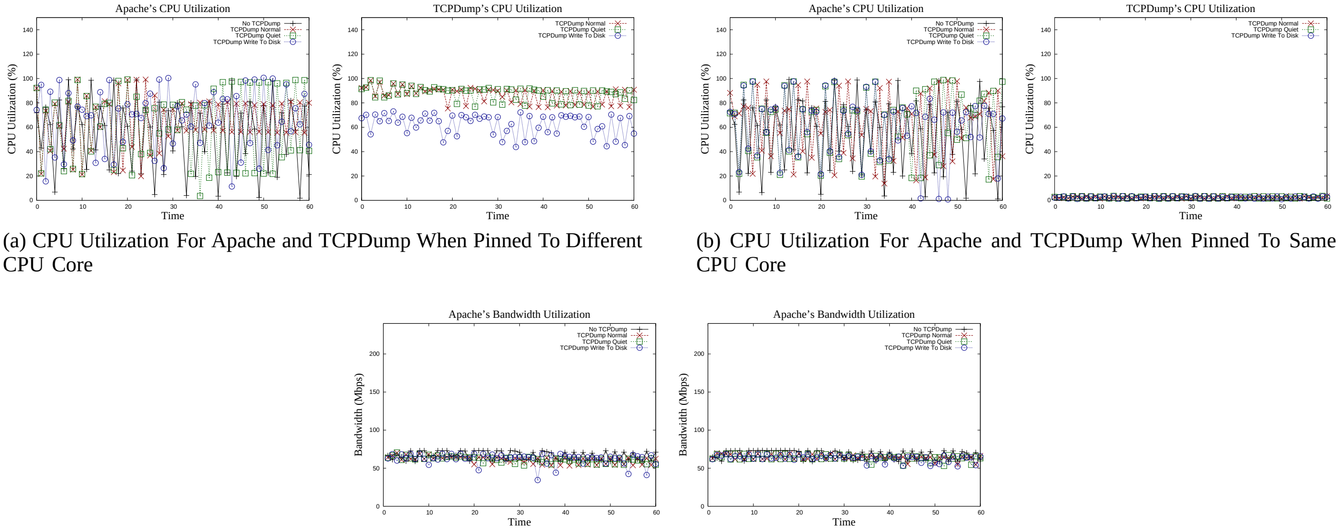
Figure 1 and Figure 2 show the CPU and bandwidth utilization for one run of the experiment, when running the HTTP_LOAD to retrieve files from the web servers.

Upon first glance, we note that the behavior of Apache and Nginx are visibly different, with Apache more prone to CPU fluctuation, while Nginx is more stable in CPU usage. This could be attributed to the fact that Apache's method of scaling with demand results in much more context switches between the various worker threads, resulting in the performance fluctuation. On the other hand, Nginx is using

[1] The way in which the packet is copied is that an internal structure, sk_buf, that holds information regarding the data packet is copied. But only the fields are copied – the packet data itself is referred by pointers in the duplicate and the original structure.

[2] We choose this model based on the scalability and smaller memory footprint than pure process-based module

[3] We also consulted the documentation on the popular prefork module and it works in similar manner.


Apache's CPU Utilization

TCPDump's CPU Utilization

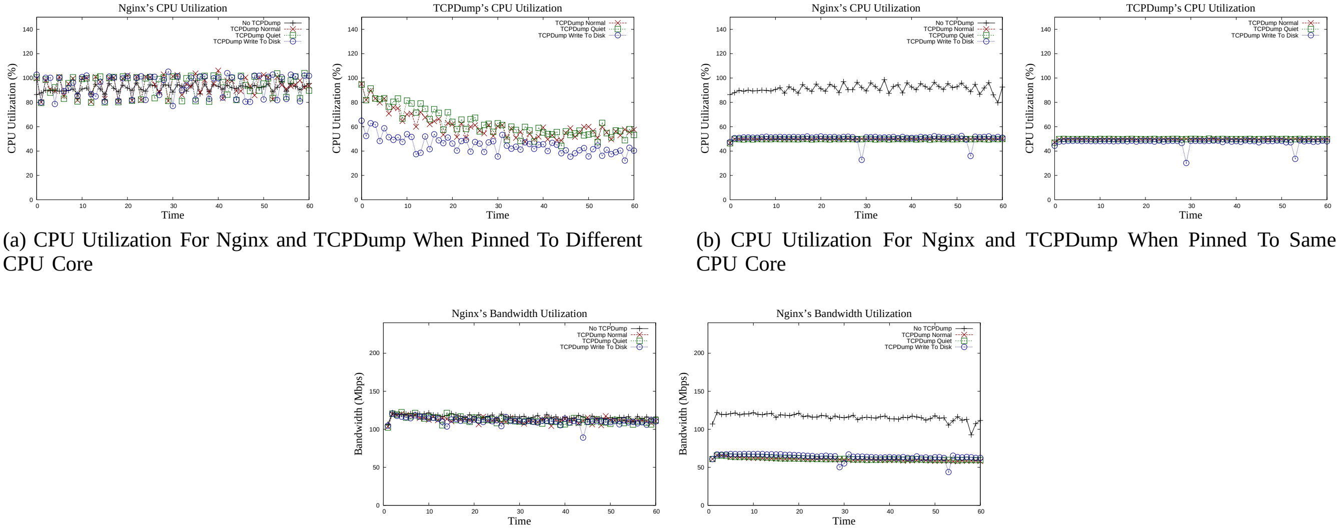
(a) CPU Utilization For Apache and TCPDump When Pinned To Different CPU Core

(b) CPU Utilization For Apache and TCPDump When Pinned To Same CPU Core

Apache's Bandwidth Utilization

(c) Bandwidth Utilization When Apache and TCPDump Are Pinned To Different (left) Or The Same CPU Core (right)

Fig. 1: Apache Results


Nginx's CPU Utilization

TCPDump's CPU Utilization

(a) CPU Utilization For Nginx and TCPDump When Pinned To Different CPU Core

(b) CPU Utilization For Nginx and TCPDump When Pinned To Same CPU Core

Nginx's Bandwidth Utilization

(c) Bandwidth Utilization When Nginx and TCPDump Are Pinned To Different (left) Or The Same CPU Core (right)

Fig. 2: Nginx Results

much less number of processes, and is able to serve requests without getting constantly interrupted.

With the web server and TCPDump pinned to different CPU core, both processes are shown to take up significant amount of CPU resource. In both types of web servers, the web server process takes up nearly one hundred percent of the process while the TCPDump process also consumes significant CPU resource. When the server and TCPDump process are bound to the same CPU core, the Apache server seems to be holding more share of the CPU resource while Nginx predictably shares the resource equally with TCPDump. In fact, TCPDump has almost no access to the CPU, using only a few percent of the CPU resource at any given time. This phenomenon can be explained by understanding the method in which Apache scales with increasing user demand. To maintain scalability, the worker MPM in Apache has a parent process that monitors and distributes incoming load. The parent process spawns a number of child process that actually serve the request, with the maximum number of child processes constrained by the ServerLimit directive (defaults to 16, which is our setting). The kernel scheduling algorithm would then try to equally distribute the available CPU resource to all active process, majority of which belongs to Apache. On the other hand, Nginx and TCPDump predictably shares the CPU resource equally, due to the fact that only one Nginx process is actively serving incoming requests. This result has significant implication to the efficiency of the web server and packet capturing process. To ensure capturing process has access to enough CPU to process the captured packet, it should be scheduled on a separate core. If the capturing process is to be co-located with the web server, care must be taken to understand the structure and behavior of the web server, to ensure the capturing process also has access to the CPU resource. It is interesting to also note that, in some cases the recorded CPU utilization is over one hundred percent. This is impossible because we confine the processes to a single core, so the maximum possible is one hundred percent. After inspecting the source code of the tool we used to collect the CPU utilization, we noticed the tool reads two pseudo-files to calculate the CPU utilization for a process, and the two reads are not atomic. We believe that this behavior, in addition to the CPU collection process running on another thread, means that there is a potential race condition, and the calculation could be slightly off. However, we note that despite the observed error, the general trend is still significant enough for us to make the above observations.

Next we look at the bandwidth utilization result. For Nginx the achieved bandwidth utilization is lower in the case when TCPDump is running and co-locating with the web server. On the other hand, the bandwidth utilization for apache is about the same as before. Comparing Figure 1c and Figure 2c with Figure 1b, Figure 2b, the difference in bandwidth usage is correlated with the difference in CPU usage: a higher CPU usage corresponding to higher bandwidth utilization, and vice versa; however the variability in the bandwidth/CPU utilization are different, as the bandwidth utilization remains more stable than CPU utilization.

While these metrics shed some light on the resource consumption and possible performance of the system, they do not explicitly tell us the performance that users can expect. To gain such an insight, we scrape the reports generated by HTTP_LOAD at each client machine at the end of the experiments. Figure 3 shows the aggregate number of fetches per second that are observed from all the clients. We note that the baseline experiment (i.e., TCPDump is not running) shows the performance of Apache and Nginx is quite good. With the presence of TCPDump, the performance of the web server varies depending on the co-location. When TCPDump is not co-located with the server process, TCPDump seems to decrease the average performance of Apache more significantly (up to 10%). However, when TCPDump co-locates with the server process on the same core, the result is more dramatic. Correlating the results in Figure 3 with Figure 1b and Figure 2b, we can see that the CPU share TCPDump has obtained is directly proportional to the decrease in the average fetches per second achievable. The significant performance degradation for the case where TCPDump and Nginx are on same core – but little performance degradation when on different cores – suggests that CPU utilization could be a major source of the web servers' performance degradation.

In summary, the experiments carried out in this section implies that CPU utilization can tell us that performance degradation has occurred, but performance degradation can still occur even if CPU utilization looks normal. In the case for Apache, even though it dominates the CPU when co-located with TCPDump, the fetches per second achievable is considerably lower; while Nginx consumes much less CPU but has similar performance degradation as Apache. However it is undeniable that the presence of TCPDump has negative performance impacts to the web servers, so care should be taken when running packet capturing process such as TCPDump, as to ensure the performance impact to the web servers is minimal. In addition, when TCPDump has equal opportunity to contend for the CPU resource it does consume a non-trivial amount, and this has performance impact for admins looking to consolidate different types of task onto a single machine. For tasks that are CPU-bound, consolidating it with machines running packet capturing processes could elongate the task completion time as well as diminishing the number of packets captured.

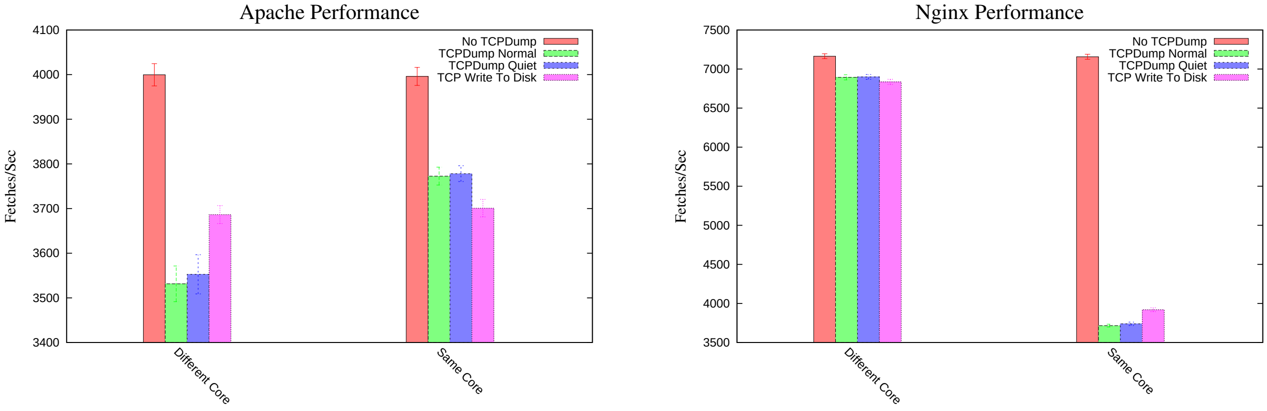
IV. Related Works

For evaluation of packet capturing tools, the closest works to this paper are those that either explicitly evaluate packet capturing performance or attempts to improve the packet capturing performance. This is because in both types of work, an evaluation of the various aspects of packet capturing tools such as CPU utilization and packets captured are usually presented. Below we briefly describe both types of work.

Deri [11] has suggested that the packet capturing process is inefficient due to overhead involved in copying the packet. The work proposes a new socket type, PF_RING, in which the packet can be copied directly from the device driver buffer to user-accessible memory, drastically reducing memory allocation and copying operations. A later work improved upon PF_RING by proposing a new architecture in which multi-core processor can be utilized to increase the monitoring capability of the system [13]. In both of these works, Deri et al. discovered that the capturing process do not handle high traffic volume well due to the memory operations from device driver to kernel and from kernel to user level, as well as sub-optimal utilization of resource available at device and kernel level.

In [14], [15], the authors investigate the performance of packet capturing tools in various software and hardware platform. The metrics investigated in these works are the packets captured [14], [15], with [14] having some emphasis on the CPU utilization and [15] focusing on the percentage of packets captured. Both works are important because they evaluate the performance of packet capturing tools using common platforms, so the valid conclusions can be drawn regarding the hardware or software stacks involved in packet capturing.

Our paper differs from these works in that we do not emphasize on the performance of the packet capturing tool, but whether the packet capturing tool affects existing applications, and if so to what degree. Even though the hardware used to host the capturing process and web server is multi-core, we only utilize one core for either the web server or the packet capturing process. This is so we could monitor the effect of co-locating the two processes, and have shown that co-location causes dramatic performance degradation to the web servers.



(a) Apache Server's Performance

(b) Nginx Web Server's Performance

Fig. 3: Web server performance in fetches/second averaged over 24 runs, with 95% confidence interval shown

V. Conclusion And Future Works

In this paper we examine the performance of web servers in the presence of packet capturing process. We find that CPU sharing is directly proportional to the performance degradation experienced by the web server, and separating the two processes onto different cores still has some performance impact on the web server.

This work is a good start, but more environments can be considered:

- **Serving larger web pages**: We need to repeat the experiments for the case when web servers are serving larger web pages. This would make the web server more I/O bound, and having the capturing process write to disk should create another venue for resource contention.
- **Serving dynamic pages**: In this paper we have looked at the case when web servers host static pages, dynamic page would put more CPU demand on the server, and the performance impact of such needs to be investigated.
- **Caching pages**: When serving static pages, the web page can be cached in memory, thus avoiding the disk completely. More experiments should be performed to investigate the effect of such a strategy.
- **Monitoring technology**: In this work we do not take advantage of the prototypes made available from prior research works (e.g., PF_RING), other techniques (e.g., sampling), or other types of packet capturing tools (e.g., dumpcap), future work will investigate these varieties.

We believe this work is the first step towards thoroughly understanding the behavior of co-locating capturing process and web servers. From these experiments, we can understand how to best capture packets when the capturing process has to be co-located with the on-line service, and whether new techniques can be applied to perform network measurement.

References

[1] L. Popa, B.-G. Chun, I. Stoica, J. Chandrashekar, and N. Taft, "Macroscope: end-point approach to networked application dependency discovery," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, ser. CoNEXT '09. New York, NY, USA: ACM, 2009, pp. 229–240. [Online]. Available: http://doi.acm.org/10.1145/1658939.1658966

[2] X. Chen, M. Zhang, Z. M. Mao, and P. Bahl, "Automating network application dependency discovery: experiences, limitations, and new solutions," in *Proceedings of the 8th USENIX conference on Operating systems design and implementation*, ser. OSDI'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 117–130. [Online]. Available: http://portal.acm.org/citation.cfm?id=1855741.1855750

[3] P. Bahl, R. Chandra, A. Greenberg, S. Kandula, D. A. Maltz, and M. Zhang, "Towards highly reliable enterprise network services via inference of multi-level dependencies," in *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, ser. SIGCOMM '07. New York, NY, USA: ACM, 2007, pp. 13–24. [Online]. Available: http://doi.acm.org/10.1145/1282380.1282383

[4] G. Insolvibile, "Inside the Linux Packet Filter." [Online]. Available: http://www.linuxjournal.com/article/4852

[5] ——, "Inside the Linux Packet Filter, Part II." [Online]. Available: http://www.linuxjournal.com/article/5617

[6] Apache, "Apache Software Foundation." [Online]. Available: http://www.apache.org/

[7] Nginx, "Nginx." [Online]. Available: http://wiki.nginx.org/Main

[8] NetCraft, "February 2011 Web Server Survey," 2011. [Online]. Available: http://news.netcraft.com/archives/2011/02/15/february-2011-web-server-survey.html

[9] I. F. Haddad, "Apache 2.0: The Internals of the New, Improved." [Online]. Available: http://www.linuxjournal.com/article/4559

[10] D. Kegel, "The C10K problem," 2006. [Online]. Available: http://www.kegel.com/c10k.html

[11] L. Deri, N. S. P. A, V. D. B. Km, and L. L. Figuretta, "Improving passive packet capture: Beyond device polling," in *In Proceedings of SANE 2004*, 2004.

[12] A. Laboratories, "http_load - Multiprocessing HTTP Test Client." [Online]. Available: http://www.acme.com/software/http_load/

[13] F. Fusco and L. Deri, "High speed network traffic analysis with commodity multi-core systems," in *Proceedings of the 10th annual conference on Internet measurement*, ser. IMC '10. New York, NY, USA: ACM, 2010, pp. 218–224. [Online]. Available: http://doi.acm.org/10.1145/1879141.1879169

[14] F. Schneider and J. Wallerich, "Performance evaluation of packet capturing systems for high-speed networks," in *Proceedings of the 2005 ACM conference on Emerging network experiment and technology*, ser. CoNEXT '05. New York, NY, USA: ACM, 2005, pp. 284–285. [Online]. Available: http://doi.acm.org/10.1145/1095921.1095982

[15] L. Braun, A. Didebulidze, N. Kammenhuber, and G. Carle, "Comparing and improving current packet capturing solutions based on commodity hardware," in *Proceedings of the 10th annual conference on Internet measurement*, ser. IMC '10. New York, NY, USA: ACM, 2010, pp. 206–217. [Online]. Available: http://doi.acm.org/10.1145/1879141.1879168