Improving NIDS Performance Through Hardware-based Connection Filtering

Vikas Garg, Vinod Yegneswaran and Paul Barford
Dept. of Computer Sciences, University of Wisconsin – Madison
{vikas,vinod,pb}@cs.wisc.edu

Abstract—**Traffic volume and diversity can have a significant impact on the ability of network intrusion detection systems (NIDS) to report malicious activity accurately. Based on the observation that a great deal of traffic is, in fact, not important to accurate attack identification, we investigate *connection filtering* as a method for improving the performance of NIDS. We describe three different classes of connection filters that were developed to explore the design space and trade off's in load reduction versus alarm rates. We implement instances of each filter class on a network processor that can be used with any NIDS that runs on commodity hardware, and evaluate the impact of each filter in a series of laboratory-based tests. First, we establish an idealized maximum performance by using *static connection filters* for all benign traffic. Next, we show that *volume sensitive random connection filters* can improve performance significantly with respect to alarm rates under heavy traffic load. Finally, we show that *dynamic connection filters* that attempt to infer benign traffic can improve performance almost to the level of idealized static filters. These results underscore the potential for hardware-based connection filtering as an effective means for improving the performance of NIDS.**

I. Introduction

Networks are under constant assault from a variety of threats including worms, viruses, DoS attacks and malicious scans. Over the past several years, the sophistication and diversity of these threats has steadily increased along with the overall volume of malicious traffic making the task of protecting networks increasingly difficult. This speaks directly to the need for new tools and techniques for protecting networks that can address the challenges of scale and diversity in malicious traffic effectively.

Network intrusion detection systems (NIDS) are a critical component of comprehensive network security architectures. NIDS monitor traffic, typically near network borders, and generate alarms when a specific attack profile is recognized by either statistical or signature-based methods. Ideally, NIDS generate *accurate* alarms – namely no false alarms are raised (*i.e.,* no false positives), and all attacks are recognized (*i.e.,* no false negatives). Prior work has shown that the ability of a NIDS to operate effectively depends, to a large extent, on how the system manages resources in the face of dynamic, high volume packet traffic [5], [17].

Our objective is to improve performance of NIDS running on commodity hardware. We define performance in terms of the ability of a NIDS to monitor a given traffic stream without becoming overloaded as evidenced by excessive packet loss and corresponding false alarms similar to [5], [17]. Our approach is based on the observation that a great deal of the traffic monitored by NIDS is either benign or redundant malicious traffic and therefore could be filtered (*i.e.,* blocked before it reaches the NIDS) without having a negative impact on alarm accuracy. If this can be done, then we would expect that the NIDS would have lower computation demands, maintain less state and should therefore perform better for a given traffic volume. While this general observation has been made in the past, our work is distinguished by both our filter designs and our implementation approach.

We propose *hardware-based connection filters* as a mechanism to identify and discard packets from specific connections before they are passed on to the NIDS. In this study, we develop and evaluate the effectiveness of three classes of connection filters. The first are static filters that use simple predefined rules to decide which traffic to pass to the NIDS. If we had *a priori* knowledge of all benign connections then static filters could be used to establish an upper limit on the effectiveness of connection filtering. The second filter class are those that discards connections at random *i.e.,* without the need for significant state maintenance in the filter. Our conjecture is that if a large percentage of traffic is benign, then random discard algorithms have the potential to prune traffic without affecting alarm accuracy significantly. The third filter class are those that attempt to actively distinguish benign and malicious connections based on observed scanning behavior. Using this more stateful approach, we hope to improve upon random filtering by reducing false negative alarms. We describe instances of each of these filters in detail in Section III.

At first glance, a reasonable question about our approach might be how to isolate a filtering process that is potentially resource intensive from the NIDS running on the same host system. Indeed, the other NIDS traffic filtering systems that we are aware of run on the host at the same time and therefore contend for resources with the detection processes. We address this issue by enhancing the host system with a network processor (NP). Network processors are systems implemented on a daughter card that provide a high speed network interface and programmable packet processing capability at or near line rate. The benefits of our implementation choice are the following:

- NP-based connection filters are *NIDS independent*. They can be used with *any* NIDS that runs on commodity hardware.
- NP-based connection filters are *exploit independent*.
- NPs offer a flexible high performance system well suited to filtering.

While NPs have been used for a variety of applications including intrusion detection (*e.g.,* [4]), to the best of our knowledge, we are the first to explore the use of NPs in this context. We implemented instances of each connection filter type on an Intel IXP2400 network processor system, and then tested their utility in a series of controlled laboratory experiments using the Bro NIDS [14] running on a commodity PC. Our experiments

evaluated false alarm rates over a range of benign and malicious traffic volumes with each filter. Using the static filter we establish an upper bound on the capabilities of connection filtering showing that they can improve NIDS effectiveness by as much as 73%. Next, we find that our volume-sensitive random filter improve NIDS effectiveness to within 10% of the maximum under heavy traffic load. Finally, we show that our dynamic filter can improve NIDS effectiveness to within 17% of the maximum but with a significantly different CPU utilization profile than the random filter. These results indicate the great potential of connection filtering in NIDS in general and the use of NPs for this function in particular.

II. Related Work

Over the past several years there has been an increasing number of empirical studies of the characteristics of malicious network traffic. In a series of papers, Moore *et al.* have provided details on the major worm outbreaks [11], [12]. Recent work based on the use of network honeypots has shed light on the characteristics of unwanted network traffic, most of which is malicious [13]. A detailed treatment of intrusion traffic based on a large set of logs provided by Dshield.org is given in [20]. All of these studies speak to the challenges NIDS face in terms of traffic dynamics and loads.

The problem of evaluating the design and improving performance in network intrusion detection systems has been addressed in a number of studies. Those that are most relevant to our work include papers that deal with problems related to the capacity and resource management in NIDS including [7], [9], [15], [17]. Of particular relevance is the study by Dreger *et al.* in [5] which reports real world experiences with the Bro. That paper highlights the particular resource management problems NIDS face under high traffic volume. It also provides an evaluation of two mechanisms for controlling packet load that are in the same spirit as our connection filters. However, their filter operation is quite different from ours, and their implementation is kernel-based.

The idea of using hardware-based systems to improve NIDS performance has also been investigated over the past several years. The bulk of this work has been focused on developing special purpose systems using, for example, FPGAs. The design approach of most of these systems is to implement a subset of the attack identification capability found in a standard NIDS [4], [18]. It is worth noting that the authors in [4] also test an implementation of their system on an Intel IXP1200 - the precursor to the IXP2400 [1] used in our study. This work differs from ours in that we are only focused on developing connection filtering capability in hardware. A study that is perhaps more closely related to our own is reported in [3]. The authors propose *early filtering* as a method for improving NIDS performance. Their filters, however, are based on static Snort [2] rules, their implementation is kernel-based and their tests are run offline.

III. Connection Filters

A. Design

Our approach to enhancing the performance of NIDS is to filter connections before they are passed on to the NIDS for evaluation. Ideally, the filtering objective is to prune all benign traffic as well as redundant malicious traffic. The challenge then becomes to construct a filter with the ability to classify traffic accurately as either benign or redundant malicious. In essence, this is the dual of the challenge faced by NIDS *i.e.,* to recognize only the non-redundant malicious component of traffic. We propose three different classes of connection filters as follows:

1. Static Filtering: The first strategy that we propose is *static filtering*. The idea is straight forward; if a particular connection is known to be benign or redundant malicious, then do not forward packets from that connection to the NIDS. Conceptually, static filters have the benefit of simplicity, and with perfect knowledge of all benign and redundant malicious connections, this filter should offer the maximum possible benefit to the NIDS. While this kind of omniscience is clearly unreasonable in practice, the static filter is useful for two reasons. First, it enables us to establish a high water mark for maximum possible performance improvement by connection filtering thereby enabling other filters to be evaluated more objectively. Second, static filters are, in fact, already commonly employed in NIDS thereby enabling us to understand the potential performance improvement of removing this activity from the host CPU. Our primary intention in developing a static filter is to investigate the former.

For our laboratory-based experiments, we can, in fact, have complete knowledge of all benign connections. Therefore, when a packet reaches this filter, a static bitmap of source IP addresses is consulted to decide whether to forward the packet to the NIDS (*i.e.,* it is unequivocally from a malicious source) or to drop it (*i.e.,* it is unequivocally from a benign source). Our static filter implementation does not consider redundant malicious traffic although this would not be a difficult extension.

2. Random Filtering: The second filtering strategy that we propose is *random filtering*. The idea is that if a large percentage of traffic is benign, then a connection filtering methods based only minimal aggregate state should be effective. Uniform random sampling represents the most conceptually simple instance of this approach. The potential drawback of random filtering is that as the volume or mix of of malicious traffic grows, there is likely to be a corresponding increase in false negative alarms for malicious connections that were filtered.

Consistent with our basic observations of NIDS performance, we hypothesize that adapting the connection filtering rate to the observed packet volume could reduce the number of false alarms caused by a random filter. We supplement our instance of the uniform random filter with load adaptation capability that is modeled after the Random Early Detection method for controlling congestion in router queues [6]. In our implementation, when traffic volume is below a threshold α, no connections are filtered. As traffic volume rises above that threshold, the probability of connection filtering increases linearly in accordance with volume. Finally, an upper threshold β on forwarded connections can be fixed to prevent a particular NIDS from going into overload.

3. Dynamic Filtering: The third filtering strategy that we propose is *dynamic filtering*. The idea is that if some measure of insight could be gleaned as to whether a particular host is benign or malicious, then perhaps more informed connection filter-

ing choices could be made. From one perspective, the dynamic filter is analogous to the static filter in the sense that it selects connections to deny based explicitly on packet header information (as opposed to random filtering). However, the set of IP addresses used in filtering changes based on inference about the nature of the traffic. We investigate two instances of dynamic filters: *source-level* and *target-port-level*.

Under source-level filtering, the past behavior of specific sources is used to judge their "trustworthiness". The method for establishing trustworthiness is to count the number of successful and unsuccessful connections made by each observed source IP address. This heuristic is based on the expectation that malicious sources, since they pick targets randomly, experience a larger number of failed connections [8]. In our prototype implementation, connections from all source IP addresses begin with a "suspicious" tag and a connection attempt counter τ set to a default value. Each unsuccessful connection attempt (*e.g.,* when no SYN-ACK is observed or a RST is observed) results in τ being decremented by one. Likewise, each successful connection attempt (*e.g.,* when a SYN-ACK or ICMP echo-response is observed) results in τ being incremented by one. When τ goes beyond a threshold ϕ, then that source IP address gets tagged as "benign" (it's tag will change back to suspicious if τ falls back below ϕ) The dynamic filter forwards all connection from source IP addresses tagged as suspicious to the NIDS. This method is clearly susceptible to source address spoofing, and we intend to address this issue in future work.

Target-port-level filtering is a straightforward extension of the source-level strategy. It computes the "malice" of each target port again using successful versus unsuccessful connections as the metric. In this case, if a port is deemed to be receiving exclusively benign connections, then further connections to this port are filtered from the NIDS. These filters are meant to improve adaptability and resilience to false alarms during surges in benign traffic. We do not implement dynamic port filtering in this study and leave that for future work.

B. Discussion

While the connection filters described above provide an initial framework for considering the possibilities of improving NIDS performance (as will be seen in the following section), there are many other possible designs. For example, a simple extension would be to add volume-aware adaptivity to dynamic filters. Another possibility would be to appeal to stratified sampling theory to improve the performance of random filters. A further possibility would be to dynamically adjust filters based on receiving feedback from the host NIDS.

C. Network Processor Implementation

An important aspect of our overall approach to connection filtering is to off-load the entire activity from the host CPU. While connection filters could be run simultaneously on the host system , off load to a network processor in theory enables maximum resource (CPU and memory) utilization by the NIDS. Our NP-based approach has the further benefit of enabling portable use with any NIDS that runs on a commodity host.

We developed our network processor-based connection filters using the ENP2611 daughter card from Radisys. This system has an embedded Intel IXP2400 processor. The IXP2400 has 8 micro engines for data plane functions at line rate, and an ARM processor for control plane functions and for communicating with the host. The system has 256MB of DDR SDRAM, 8MB of QDR SRAM, and uses the PCI Bus to interface with the host machine. It has three Gigabit Ethernet ports for data and one 100Mbps Ethernet port for management and development tasks.

Each of the connection filters was implemented on the micro engines in the IXP2400 using IXP assembly language. The average size of each filter was several hundred lines, and they were all implemented using multiple threads. The application is broken into three distinct stages: *receiving, filtering and forwarding*. The receive and filter stages are implemented on the micro engines, while the forwarding stage (up to the host NIDS) is implemented on the ARM processor since it controls access to the PCI bus interface. We use pipelining between the three stages to improve performance. Synchronization across stages is enabled through scratch-pad rings which support atomic data transfers between micro engines.

The host system used in all of our tests was a single CPU system configured with a 2.8 GHz Intel P4/Xeon processor, 1MB L2 cache and 2GB of main memory running Linux Fedora 2 with the 2.6.5 kernel. The ARM processor on the IXP2400 was running Montavista Linux 3.0 with kernel 2.4.18. In tests that we ran with a network interface card instead of the IXP2400, the NIC was an IntelPro 1000 GE card.

IV. Experimental Evaluation

A. Setup and Configuration

Our experimental objective is to evaluate the impact of our hardware-based connection filters on a popular NIDS running on commodity PC. The specific focus of our tests is to measure alarm rates and CPU utilization over a range of workloads that include both benign and malicious traffic. Our evaluation was conducted in a controlled laboratory environment that enabled us to monitor all aspects of the experiments including packet loss rate on the NIDS host.

Our target NIDS is Bro-0.9a8 configured with the standard `brolite.bro` policy script. The only modification we made to the script was to disable all packet filters. While we use Bro for our experiments based on our own experience with the tool, we believe that the impact of NP-based connection filtering is likely to be similar for other open-source NIDS that run on commodity hardware such as Snort [2].

We generated benign traffic using 6 PCs running Harpoon [16]. Each machine was configured with 16 class C subnets for a total of 24,576 source/destination IP addresses. While Harpoon is useful for generating traffic flows that approximate what is seen at a live router, it does not include the capability to reproduce diverse, representative *packet content* which is important in NIDS tests. We enhanced Harpoon by enabling it to include benign content from the Darpa99 data set [10].

We generated malicious traffic using 2 PCs running MACE [17]. We use IP-aliasing to emulate source and destination IP addresses from 32 different class C subnets (16 subnets for client IP addresses and 16 subnets for server IP addresses).

Host Based					Network Based
Application level			Transport level		
Worms	Backdoors	DoS	Fragmentation	Other DoS	
Welchia Nimda CodeRed2 Blaster Dameware Sasser	mydoom sdbot	winnuke	rose teardrop1 teardrop2 bonk nestea oshare	synflood pod land jolt	smurf fraggle

TABLE I TAXONOMY OF MACE ATTACK TYPES USED IN EXPERIMENTS.

	Malicious Traffic		Combined Traffic	
	Mbps	pps	Mbps	pps
Test-30	3	700	30	8,000
Test-60	6	1,400	60	16,000
Test-90	9	2,100	90	24,000
Test-120	12	2,800	120	32,000

TABLE II AVERAGE TEST TRAFFIC LOAD LEVELS.

We used the iSink system's active response capability to send response packets to malicious MACE traffic [19]. This enables bidirectional communication required to accurately recreate certain exploits. The malicious traffic generated by MACE was a randomly selected mixture of a set of 21 typical attacks listed in Table I. We do not claim that this distribution of attacks is representative of any particular network. However, all of these attacks are commonly seen in today's traffic and we believe that this set exercises a NIDS in sufficiently diverse ways.

The aggregate traffic mixture used in all experiments was comprised of 90% benign and 10% malicious by volume on average. We choose this mixture as a convention for our experiments with the understanding that the aggregate traffic in any given network is likely to be highly variable, and that the potential benefit of our filters is largely dependent on the amount of benign traffic.

We ran the tests using 4 different average traffic load levels to evaluate connection filter performance as shown in Table II. It should be noted that loads varied dynamically during tests per our Harpoon and MACE configurations. The minimum and maximum average load levels were determined through trial and error to demonstrate in a parsimonious way the range of performance with and without connection filters.

In our experimental setup, all the traffic generator systems were connected using a single VLAN on a Cisco Catalyst 3750 switch. We used port mirroring to forward all of the traffic to two target systems. One of these systems ran the NIDS and was configured with both the standard NIC and IXP2400. The other target system was configured to take a full packet trace during all tests using `tcpdump`. These traces were then used for off-line validation of our results.

We conducted the six experiments listed below at each of the four traffic levels. Each experiment was run for 10 minutes during which MACE logged the malicious attacks it initiated and Bro logged the malicious attacks it identified. Using these two logs, we evaluate *alarm effectiveness* which is defined as the ratio of the number of true alarms generated by Bro to the total number of malicious attacks generated by MACE (*i.e.,* a measure of false negatives where a value of 1 indicates there were no false negatives). Malicious attacks that are filtered during tests will result in a decrease in alarm effectiveness. It is important to note that single instances of several of the attacks generated by MACE result in multiple alarms in Bro. Our alarm effectiveness calculation normalizes the Bro alarm count based on off-line, atomic evaluation of each MACE attack type. Despite this, different combinations of attacks that occur during experiments occasionally suppress expected alarms. This leads to effectiveness levels slightly less than 100% even though Bro is accurately identifying all attacks as will be seen below. We also track CPU utilization on the NIDS host and the percentage of packets processed by Bro during all tests.

1. In the first experiment, we generate only malicious traffic from MACE that Bro monitors via the host PC's NIC. This experiment is meant to establish the maximum effectiveness of benign connection filtering for each of the four traffic levels.
2. In the second experiment, we generate combined malicious (MACE) and benign (Harpoon) traffic that Bro monitors via the host PC's NIC. This experiment is meant to establish the baseline performance of the NIDS in terms of effectiveness, CPU utilization and packet processing against which we will compare the NP-based connection filters.
3. In the third experiment, we generate combined malicious/benign traffic that Bro monitors via the NP with no filtering enabled. This experiment enables us to compare the performance of the NP versus the NIC (from experiment #2) in terms of simple packet forwarding to the NIDS. This experiment is important for validating the receiving and forwarding components of our NP implementation.
4. In the fourth experiment, we generate combined malicious/benign traffic that Bro monitors via the NP with static filtering enabled. If configured to block all benign connections from Harpoon, this experiment enables us to assess this simplest filter implementation by comparing results with experiment #1.
5. In the fifth experiment, we generate combined malicious/benign traffic that Bro monitors via the NP with the RED-like random connection filter enabled. Our prototype filter was quite simple. We calculated the packet rate and adjusted the filter rate on 30 second intervals. The α threshold was set to 15,000 pps with random connection filtering at 12.5%, 25% and 50% for rates greater than 15K pps, 30K pps and 45K pps respectively. A more sophisticated implementation and a sensitivity analysis vis-á-vis thresholds are subjects for future work.
6. In the sixth experiment, we generate combined malicious/benign traffic that Bro monitors via the NP with the dynamic connection filter enabled. We configure τ to be 128, and ϕ to be 150 for all experiments. A more sophisticated implementation and a sensitivity analysis vis-á-vis the thresholds are subjects for future work.

B. Results

Experiment #1: Figure 1 shows the results for malicious only traffic transmitted to the NIDS via the NIC. At low traffic volumes, Bro is able to process all the packets, CPU utilization is modest, and the alarm effectiveness is close to 100%. As noted above, the fact that it is not 100% is an artifact of how Bro reports alarms for combined attacks, and this effectiveness level can be considered maximum. At the Test-90 load level (9 Mbps of malicious traffic) the system nears it performance limits as

the CPU utilization goes to 100%. At this level, the effectiveness is still maximum since the system has not started dropping packets yet. At the Test-120 load level (12 Mbps of malicious traffic), the NIDS is no longer able to keep up with the offered load as evidenced by the decrease in packets processed. The figure shows that the NIDS processed only 82% of the incoming packets and the alarm effectiveness goes down by about 11% to 85%.

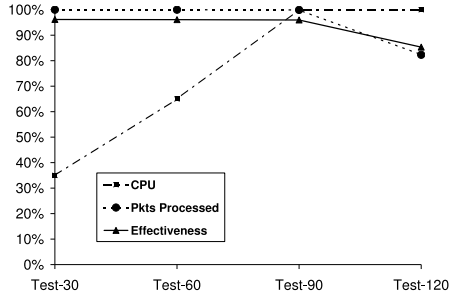

Fig. 1. NIDS performance with only malicious traffic transmitted via NIC

Experiment #2: Figure 2 shows the results for the combined malicious/benign traffic transmitted to the NIDS via the NIC. The graph demonstrates that the addition of benign traffic has a significant impact on the performance of the NIDS even under moderate overall traffic levels. At the Test-30 load level, the NIDS is able to process all the packets. However, even at the Test-60 load level, the NIDS drops a significant number of packets. At highest Test-120 load level, the NIDS is able to process only 10% of the packets and effectiveness goes down to 10%.

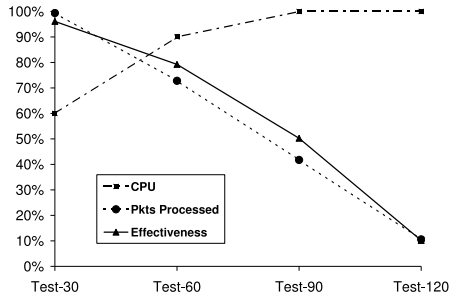

Fig. 2. NIDS performance with malicious/benign traffic transmitted via NIC

Experiment #3: Figure 3 shows the results for combined traffic transmitted to the NIDS via the NP without any filtering enabled. As expected, the NIDS performance in this experiment is quite similar to that in experiment #2 in which the NIDS receives packets via the the NIC. The primary difference is that the CPU utilization is a bit higher at lower load levels. This is due to the fact that the driver used to interface the host with the IXP2400 is an unmodified prototype implementation that was shipped with the Radisys board. We feel that a closer examination of the driver will lead to modifications resulting in a reduction in CPU utilization.

Experiment #4: Figure 4 shows the results for combined traffic using the NP-based static filter. In this experiment, the static filter is configured to block all benign connections based on source IP addresses, thus only the malicious traffic is sent from the NP to the NIDS. As expected, the performance is nearly the same as the results of experiment #1 where only malicious traffic is used. It should be noted that in these tests, the NIDS is processing only a fraction (about 8%) of the total incoming traffic and that there is no discernible impact on effectiveness.

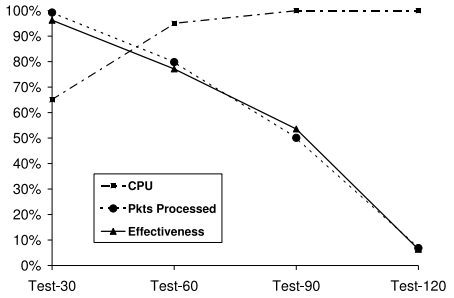

Fig. 3. NIDS performance with malicious/benign traffic transmitted via NP with no connection filter

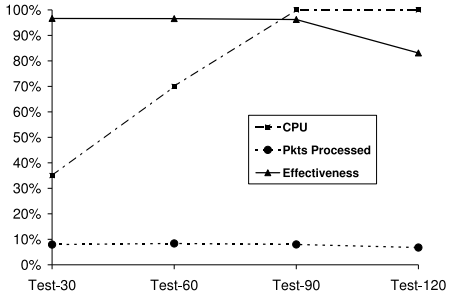

Fig. 4. NIDS performance with malicious/benign traffic transmitted via NP with static connection filtering

Experiment #5: Figure 5 shows the results for combined traffic using RED-like random connection filtering. At the Test-30 load level almost all of the packets are getting forwarded to the NIDS and the performance is similar to that of the static filter. The only obvious difference is that the CPU utilization is higher since the NIDS is processing a lot more packets. As the load level increases, the random connection filter blocks more traffic resulting in a reduction in the overall fraction of packets processed. However, alarm effectiveness stays very high even at the Test-120 load level. We were quite surprised by the capability of this filter.

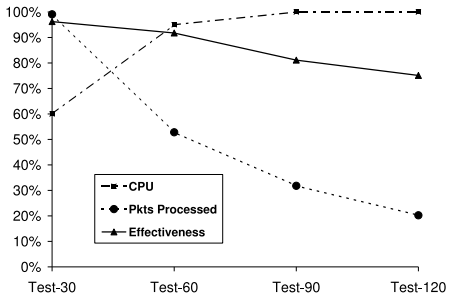

Fig. 5. NIDS performance with malicious/benign traffic transmitted via NP with RED-like random connection filtering

Experiment #6: Figure 6 shows the results for combined traffic using dynamic connection filtering. The dynamic filter is able to identify and drop almost 90% of the traffic for all tests while improving the effectiveness to within 17% of the maximum at high load levels. It should be noted that at the low Test-30 load level, the effectiveness is less than that for the NIC/NP baselines.

This is due to the fact that at the low load level, the unfiltered NIDS is able to process *all* packets, but the dynamic filter still drops connections. This suggests that an adaptive dynamic filter based on current traffic levels may be quite effective.

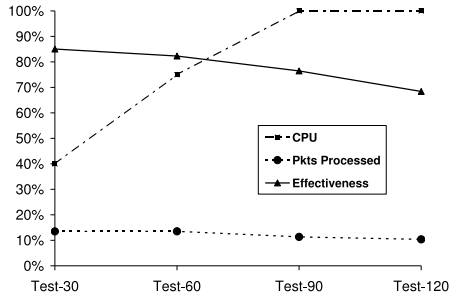

Fig. 6. NIDS performance with malicious/benign traffic transmitted via NP with dynamic filtering

V. Conclusions

Intrusion detection systems are fundamental components in network security architectures. While they have been widely studied, there remain important challenges in managing resources in face of diverse, dynamic traffic. Perhaps the most common method for addressing NIDS resource demands is to push subsets of pattern recognition functionality into hardware-based systems such as NPs or FPGAs that can inspect packets a line rate. While this approach is not without utility, it is still fraught with the basic challenges of intrusion detection not the least of which is dealing with non-malicious traffic that may still be passed up to the host CPU. This paper makes a case for developing hardware-based connection filters that complement NIDS operation on a host system.

We define three classes of *NIDS-independent* connection filters as a framework for exploring the design space. Our interest is in developing filters that minimize the false alarms that they cause and at the same time require minimal computation and state maintenance. Static filters have the lowest resource demands, and while useful for establishing baseline performance in our experimental evaluation, are likely to be ineffective in practice. Random filters have higher resource demands but offer the possibility of being much more effective than static filters in practice. Dynamic filters have perhaps the highest resource demands but offer the potential to be the most effective in practice.

We evaluate prototype implementations of instances of each of the three classes of connection filters built on the IXP2400 network processor system through series of laboratory-based experiments. While our evaluation is limited, the results are quite encouraging. For the specified traffic mix and experimental system configuration, the prototype demonstrates a factor of two improvement (from 6 Mbps to 12 Mbps) in realized NIDS scalability using hardware-based connection filtering. Specifically, idealized static filters have the potential to improve effectiveness by 75% (performance target for maximum), while volume-sensitive random filters and connection-aware dynamic filters show improvements that are within 10% and 17% of this maximum.

While these results indicate the promise of our approach, additional filter designs and a much broader set of tests are required. We believe that many new types of of both random and dynamic filters can be developed as well as possibly new classes of filters that we have not yet considered. Further, while lab-based experiments are useful for gaining insight on filter behavior and performance, a larger set of experiments with varying traffic mix and with different target systems plus *in situ* experiments are clearly required.

Acknowledgements: The authors would like to thank Intel for its support and donation of the IXP systems used in our study. This work is also supported in part by NSF grant numbers CNS-0347252, ANI-0335234 and CCR-0325653. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

References

[1] The Intel IXP 2400 Network Processor. http://developer.intel.com/-design/network/produces/npfamily/ixp2400.htm, 2005.

[2] B. Caswell and M. Roesch. The SNORT network intrusion detection system. http://www.snort.org, April 2004.

[3] I. Charitakis, K. Anagnostakis, and E. Markatos. An Active Traffic Splitter Architecture for Intrusion Detection. In *Proceedings of the ACM Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems*, Orlando, FL, October 2003.

[4] C. Clark, W. Lee, D. Schimmel, D. Contis, M. Kone, and A. Thomas. A Hardware Platform for Network Intrusion Detection and Prevention. In *Proceedings of The Workshop on Network Processors and Applications*, Madrid, Spain, October 2004.

[5] H. Dreger, A. Feldmann, V. Paxson, and R. Sommer. Operational Experiences with High-Volume Network Intrusion Detection. In *Proceedings of ACM Conference on Computer and Communications Security (CCS '04)*, Washington, D.C., October 2004.

[6] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1(4), 1993.

[7] M. Hall and K. Wiley. Capacity verification fo high speed network intrusion detection systems. In *Proceedings of Recent Advances in Intrusion Detection (RAID '02)*, 2002.

[8] J. Jung, V. Paxson, A. Berger, , and H. Balakrishnan. Fast portscan detectionusing sequential hypothesis testing. In *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, May 2004.

[9] C. Kruegel, F. Valeur, G. Vigna, and R. Kemmerer. Stateful Intrusion Detection for High-Speed Networks. In *Proceedings of IEEE Symposium on Security and Privacy, Oakland, CA, USA*, May 2005.

[10] R. Lippmann, J. Haines, D. Fried, J. Korba, and K. Das. The 1999 DARPA Off-Line Intrusion Detection Evaluation. In *Proceedings of the Symposium on Recent Advances in Intrusion Detection (RAID)*, October 2000.

[11] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. Inside the slammer worm. In *Proceedings of IEEE Security and Privacy*, June 2003.

[12] D. Moore, C. Shannon, and J. Brown. Code-Red: a case study on the spread and victims of an Internet worm. In *Proceedings of the ACM SIGCOMM Internet Measurement Workshop*, November 2002.

[13] R. Pang, V. Yegneswaran, P. Barford, V. Paxson, and L. Peterson. Characteristics of Internet Background Radiation. In *Proceedings of the ACM SIGCOMM Internet Measurement Conference*, 2004.

[14] V. Paxson. Bro: A system for detecting network intruders in real time. *Computer Networks*, December 1999.

[15] L. Schaelicke, T. Slabach, B. Moore, and C. Freeland. Characterizing the performance of network intrusion detection sensors. In *Proceedings of Recent Advances in Intrusion Detection (RAID '03)*, 2003.

[16] J. Sommers and P. Barford. Self-Configuring Network Traffic Generation. In *Proceedings of ACM SIGCOMM Internet Measurement Conference*, Taormina, Italy, October 2004.

[17] J. Sommers, V. Yegneswaran, and P. Barford. A Framework for Malicious Workload Generation. In *Proceedings of ACM SIGCOMM/USENIX Internet Measurement Conference*, Taormina, Italy, October 2004.

[18] H. Song and J. Lockwood. Efficient Packet Classification for Network Intrusion Detection using FPGA. In *Proceedings of the International Symposium on Field Programmable Gate Arrays*, Monterey, CA, February 2005.

[19] V. Yegneswaran, P. Barford, and D. Plonka. On the design and use of internet sinks for network abuse monitoring. In *Proceedings of Recent Advances in Intrusion Detection*, 2004.

[20] V. Yegneswaran, P. Barford, and J. Ullrich. Internet intrusions: Global characteristics and prevalence. In *Proceedings of ACM SIGMETRICS*, June 2003.