SNORT OFFLOADER: A RECONFIGURABLE HARDWARE NIDS FILTER

Haoyu Song, Todd Sproull, Mike Attig, John Lockwood

Department of Computer Science and Engineering
Washington University in St. Louis
1 Brookings Dr. St. Louis, MO, USA, 63130
{hs1, todd, mea1, lockwood}@arl.wustl.edu

ABSTRACT

Software-based Network Intrusion Detection Systems (NIDS) often fail to keep up with high-speed network links. In this paper an FPGA-based pre-filter is presented that reduces the amount of traffic sent to a software-based NIDS for inspection. Simulations using real network traces and the Snort rule set show that a pre-filter can reduce up to 90% of network traffic that would have otherwise been processed by Snort software. The projected performance enables a computer to perform real-time intrusion detection of malicious content passing over a 10Gbps network using FPGA hardware that operates with 10 Gbps of throughput and software that needs only to operate with 1 Gbps of throughput.

1. INTRODUCTION

Network Intrusion Detection Systems (NIDS) perform deep-packet inspection on packet payloads to identify, prevent, and inhibit malicious attacks over the Internet. Traditionally these systems are implemented in software. However, experiments show that even on moderate-speed networks, software alone is unable to process all traffic at the full link rate [1, 2]. Without scanning all traffic, some attacks will not be detected.

Snort [3] is an open source NIDS that uses signatures to detect abnormal network activities. With rules contributed by the network security community, the database of signatures is large and continues to grow. The latest version of Snort, version 2.3.2, contains over 2,600 rules. In Snort, more than 80% of the rules contain signatures and more than 80% of the CPU time for Snort is consumed by the string matching task alone [4]. As network traffic speeds increase, PC-based solutions cannot continue to process all traffic in real time.

Several attempts have been made to improve the system performance by migrating functionality from software to hardware. Though software is relatively slow, it is well suited to perform lightweight processing on low volumes of network data. On the other hand, fast hardware is best suited for computationally intensive processing on network traffic and can sustain much higher network throughput. To leverage the hardware's performance and the software's flexibility, a hybrid architecture is highly desired. The key observation is that the malicious packets typically count only a small portion of the background "normal" traffic, yet they need enormous efforts to figure out. By relaxing the screening criteria at the cost of moderately increasing the false alarm rate, a hardware plug-in or pre-processor can effectively filter out benign network traffic and only pass suspicious packets to a software system for complete inspection. This division between the software and hardware allows hardware to offload processing to match malicious packets that do not require complex inspection in software. The hardware requirements can be made sufficiently small to allow implementation on a moderate sized FPGA.

We have developed a hybrid system that can successful monitor an OC-48 link without any packet loss using a Xilinx VirtexE FPGA and a low-end PC running Linux. By using the latest Virtex4 FPGAs, the system is scalable to operate on 10 Gbps networks. The system can be implemented in a stand-alone box or in the network interface card (NIC).

2. RELATED WORK

Several approaches to improve the performance of NIDS have been proposed. Some software-based solutions use hybrid algorithms to perform rule matching [5, 6, 4, 7]. However, the performance of these algorithms is still insufficient for deployment in high-speed networks. To achieve higher processing throughput, computer clusters have been proposed to offload the workload of a single computer [8, 9]. The cost of the cluster remains high, however, because it requires multiple processors, a distribution network and a clustered management system.

Recently, there has been an increasing interest in implementing NIDS completely in hardware. Researchers have focused primarily on the computationally-intensive task of string matching [10, 11, 12, 13]. These efforts take advantage of the reconfigurability, parallelism, and advanced memory technology available in FPGA hardware to pro-

vide very high-speed string matching. Efficient circuits have been implemented in reconfigurable hardware that perform packet header classification [14]. Nonetheless, complete systems that provides full *stateful* inspection of traffic flows remain to be seen. The diversity of rules makes the problem difficult for a pure hardware implementation.

An attempt has been made to combine the network processor with reconfigurable hardware to leverage the software and hardware [15]. A dedicated NIDS card was shown to handle intrusion detection without involving extra computers with a throughput of about 100 Mbps.

In this paper, a hybrid intrusion detection system is examined. A reconfigurable hardware module offloads most of the processing from a software-based NIDS, allowing the system to scale to higher network link rates.

3. ARCHITECTURE

The overall design principle for this architecture is the use of packet filtering techniques. It is typically easier to identify a packet that does not match any rule than to ensure it matches at least one. Therefore, the pre-filter is used as a mechanism to quickly check for a necessary (but not sufficient) condition for an approximate match to occur. If the packet does not satisfy the necessary condition, it is no longer processed, leaving more time for the expensive algorithms in software to process packets that match the filter. The amount of the traffic that needs to be processed by software can be significantly reduced by our hardware filter. Moreover, the remaining software component that runs on a microprocessor could also be migrated into an embedded processor, such as the Power PC in the Xilinx V2Pro or Virtex4. The high-level architecture for our system is shown in Figure 1.

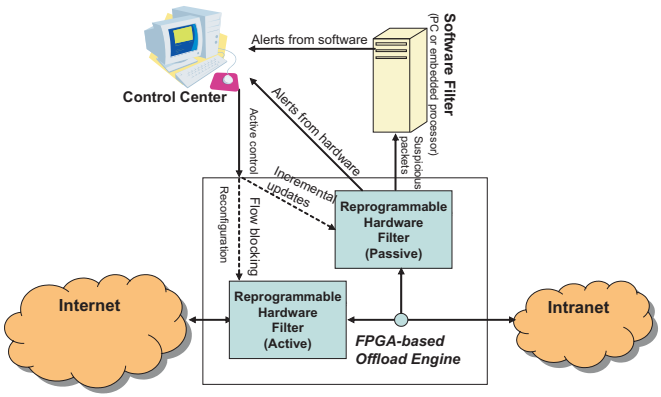

Fig. 1. System Architecture

The FPGA-based NIDS Offload Engine sits between the internal network and external network to monitor all pass-through traffic. The control center dynamically reconfigures the hardware through the network to update the filter set. The active filter sits on the data path and is used to block some flows based on the packet header. The passive filter monitors both the packet header and payload and passes the suspect packets to software. The software generates alert messages to the control center once any rule is matched. The hardware is also responsible for header-only rule matching and sends corresponding alerts directly to the control center.

The effectiveness of the passive filter can be measured by the fraction of traffic filtered in hardware to the total amount of traffic sent through the network. The amount of traffic that does not have to be inspected by software should be as large as possible. At the same time, it is critical that any real attacks should be preserved by the filter. These opposing requirements form the major challenge faced in the design of the system. First, pre-filtering packets based solely on the packet header is not desirable because some wildcard header rules match all packets. It is also not desirable to solely examine the payload. The reasons are two-fold: some rules contain only a header specification and some strings are so common that they appear in benign packets. Therefore, it is necessary to process both the header information while also scanning the payload of the packet.

The hardware filter is targeted to operate in a Xilinx XCV2000E. The design runs faster and occupies less space on the newer FPGA devices, such as the Virtex4. This design leverages previous work, as described below, and adds many new features. The hardware architecture has the following key components:

Layered Protocol Wrappers The TCP processor wrappers handle the TCP/IP packet header processing and maintain the flow states [16, 17]. The wrapper guarantees correct packet forwarding and provides flow-level monitoring.

Packet Header Classification A high performance embedded packet classification circuit implemented in reconfigurable hardware was presented in [14]. Analysis of the Snort rules showed that there are less than 300 header rules if only the standard 5-tuple header fields (i.e. source IP address, destination IP address, protocol, source port and destination port) are used. This system adds support for matching other header fields like TCP flags in order to reduce the number of packets that are processed in software. Experiments show this is crucial since some rules largely depend on these header bits to identify intrusions while other specifications are quite general. Our system classifies packets based on the 5-tuple header fields and then matches other header fields using combinational logic. The classification results are a set of matched header rule IDs.

Among 300 unique header rules in Snort v2.3.2, 144 are header-only rules. If a packet matches one of these

rules, the packet can be processed entirely in hardware. Alerts are sent directly to a control center. Hence we can delete these rules from the software rule set. This step off-loads both the input traffic and the number of rules that must be processed by software.

Two-level Bloom Filters Bloom filters are used to pre-filter the packet payload. The use of Bloom filters for string matching was first proposed in [12]. In this system, the front-end Bloom filters are used without the exact match hash table. Thus, there is no need to use any off-chip memory. Besides programming the string itself, the Bloom filters used in this system also hash over a meta-string that combines the corresponding header rule ID with the string. Since the number of unique header rules is less than 256, the meta-string is only one byte longer than the original string.

Active Packet Filter For active dropping or bypassing of packets from some flows, the control center sends directives to the hardware. The directives specify the flow using the 5-tuple IP header fields.

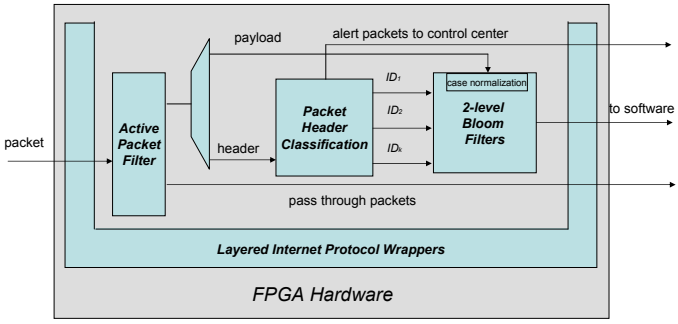

Fig. 2. Hardware Modules and Organization

Figure 2 displays the block diagram of the FPGA hardware. The packets that pass the active filter are sent out. At the same time, the header and payload of the packets are inspected. Suspect packets are sent to the software NIDS for complete inspection.

The packet header classification module and 2-level Bloom Filters module work in parallel. Initially, the packet payload is scanned by the first level Bloom filter only. When the Bloom filter reports a possible match, the string is extracted and combined with each of the header rule IDs from the header classification module. Using this meta-string, a second level Bloom filter is probed. If the second level Bloom filter reports a match, the packet is forwarded to software and no additional hardware scan is needed for this packet. Otherwise, content scanning is continued until all header rule IDs are checked and no meta-string match is found.

The approach works well for the following reasons:

1. Typically a packet will match no more than 5 header rules. In the worst case, only 5 meta-string queries are needed once the first level Bloom Filter reports a "match".

2. By checking both the string and the meta-string, the false positive rate of Bloom Filters is lowered and the amount of traffic forwarded to software is reduced.

3. Once a possible match is found, the hardware does not need to perform further inspection. The packet is forwarded to software.

4. Malicious or identified benign flows which cause excessive true matches or false matches can slow the system. This effect is reduced by configuring the active filter to either drop or bypass packets belonging to these flows.

3.1. Case Insensitive Strings

Approximately half of the strings in Snort rules are case insensitive. It is not efficient to program all possible string patterns in the Bloom filters. For example, if a string contains k case insensitive characters, 2^k distinct strings are needed to represent it. In our system, only lower-case strings are programmed into the Bloom filters. Before querying the Bloom filter using the content string, all ASCII characters are normalized to lower case. We have found that the case-insensitive only slightly increases the amount of traffic sent to software.

3.2. String Truncating and Grouping

In our system, signatures are grouped based on length, and a group is assigned to a Bloom filter. In the default configuration of Snort v2.3.2, there are 57 distinct lengths distributed from 1 byte to 107 bytes. For each length, there could be as many as 374 strings or as few as only 1 string. When comparing only the longest string for each rule (emulating the behavior of the 2 Level Bloom Filter) the lengths vary from 1 to 277 bytes. Figure 3 shows the string length distribution in Snort rules using the longest rule per string.

To make efficient use of Bloom filters, strings are clustered with different lengths in a group, and strings are truncated to the length of the shortest string in the group. Truncated strings are programmed into a single Bloom filter dedicated to this group. Using this method, only a few Bloom filters are needed to handle all the strings in the Snort rule set.

For example, when the string length threshold is set to $(t_1, t_2, ...t_k)$, all strings with length $\geq t_i$ and $< t_{i+1}$ are clustered into the group with threshold length t_i, and all strings in this group are truncated to have the same length of t_i. The string truncation can be as simple as cutting off the

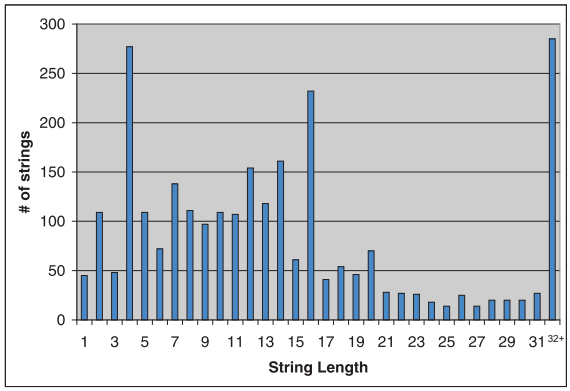

Fig. 3. String Length Distribution in Snort V2.3.2 using the longest string per rule

tail. Alternatively, analysis can be performed to select the most unique sub string to improve the system performance.

Through analysis of the rule set, we find that the average length of all the strings in Snort is less than 13. We also find that in order to make the pre-filter effective, searches for strings of length 1 and 2 must be included. The other thresholds are determined through simulation by considering performance and resource usage. A finer granularity can be achieved to lower the false positive rate. Note that the actual number of strings in each Bloom filter is equal to twice the number of signatures because of the use of meta-strings. The Bloom filter can be configured to have an extremely low false positive rate by using more block RAMs. For the strings of 1-byte long, no Bloom filter is used. Instead, a 256-entry look-up table is maintained to directly map the string. In this case, the meta-strings are 2-bytes long and programmed into a Bloom filter.

4. EVALUATION

In order to evaluate the effectiveness of hardware pre-filtering, a modified Snort rule set was generated. This circuit performed header processing and fixed-length content inspection in an FPGA. The modified rule set did not include regular expressions and did not allow multiple content strings per rule. For rules containing multiple strings, the longest string is used. Besides the header rule and truncated string, the modified rule also includes two header options: "ip_proto" and "flags". The ip_proto option checks the IP protocol field and the flags option checks for the presence of specific TCP flag bits.

Header-only rules are matched entirely in hardware, allowing for the immediate generation of alert packets. Only 144 Snort rules consist of header processing alone, making the resource requirements fairly small.

To evaluate the system with real traffic, Internet traffic from the Washington University campus was captured and processed by Snort. We analyzed all of the traffic on the 10,000+ user Washington University network, varied in bandwidth typically between 100 and 350 Mbits/sec depending on the time of day. One minute and 30 second Internet traces were collected throughout one entire day to observe variations in traffic load and types of traffic patterns. The data was sent to Snort for processing and generation of alerts.

	% of Traffic Forwarded to Software	Traffic Bandwidth (Mbits/sec)	# of BRAMs
Original rule set	0.18	0.28	-
All string lengths	12.56	21.44	880
1,2,4,8,12,16	12.96	22.19	80
1,2,4,8,12	12.96	22.19	64
1,2,4,8	13.0	22.24	48
1,2,4	13.1	22.48	32
1,2	40.0	68.42	16

Table 1. Test Results on WashU Campus Network for various String Thresholds

The first test performed evaluates the effects of different string length thresholds versus block RAM utilization, which is the critical resource of the implementation. Table 1 indicates the amount of traffic forwarded to software for different Bloom Filter string length thresholds. The rightmost column of the table indicates the number of 4K-bits block RAMs required for each test. The first row of the table indicates how the full Snort rule set would perform when implemented entirely in software and is provided for reference. The results show that by using a subset of string lengths with header rules, only a slight increase in the amount of traffic sent to software occurs while a significant savings in resource usage is achieved. However, if the number of thresholds is reduced too aggressively, such as using threshold 1 and 2 only, up to 40% of traffic has to be sent to software. The XCV2000E FPGA has a total of 160 block RAMs, thus motivating the decision to find a solution using the least amount while not introducing a considerable amount of traffic to the software. As a tradeoff, the threshold of size 1, 2, and 4 is used.

Using the threshold 1, 2, and 4, campus traffic was sampled throughout the day to demonstrate the potential savings of bandwidth sent to Snort running on a PC. Figure 4 shows the amount of traffic processed by the FPGA and the amount forwarded to software. On average, traffic is reduced by 87%. The bandwidth of traffic forwarded to software varied over the course of the day, as both the total amount of traffic varied throughout the day. We found that the largest reduction of traffic occurred at 2:00pm. Test results show that the hardware offload of Snort rule processing offers a great benefit. It allows low cost PC's to perform NIDS on

very high-speed links.

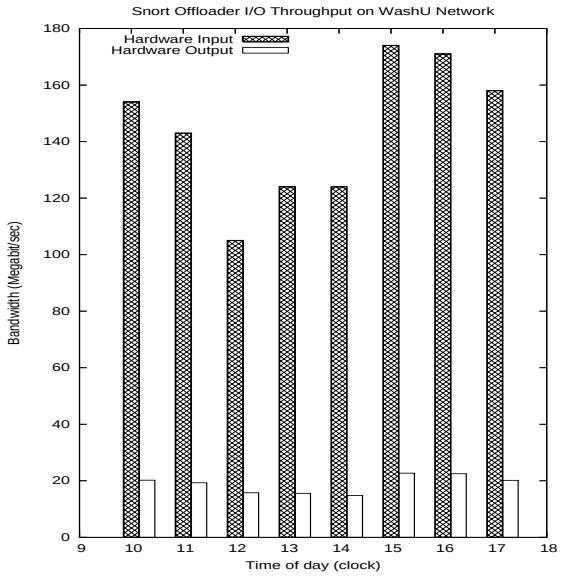

Fig. 4. Network Traffic Bandwidth to and from Hardware Offload Engine

5. IMPLEMENTATION

The entire system that processes OC-48 traffic fits into Virtex 2000E FPGA on the FPX platform. Table 2 projects how well this solution will scale for some popular network link speeds. Of the logic used in the system, the BV-TCAM of [14] required less than 10% of the logic slices in the FPGA.

	Gigabit Ethernet	OC-48	OC-192
Incoming Bandwidth	1 Gbps	2.4 Gbps	10 Gbps
Outgoing Bandwidth	131 Mbps	315 Mbps	1.3 Gbps

Table 2. Projected Throughput of Hardware Offload Engine

To process the data at the full line-rate, parallel copies of Bloom filters are used. The Bloom filters were configured to use 8 hashes to index a 16 Kbit vector. The hash is computed in 2 cycles, and the subsequent query requires 3 clock cycles. Queries can be pipelined to increase system throughput.

Each signature group Bloom filter uses 16 block RAMs and requires about 10% of the logic slices in a Xilinx Virtex 2000E. The majority of the logic is found in the hash circuitry. The largest false positive probability occurs for string group 4 with a value of 10^{-7} [12]. By using two-level Bloom filters, the false positive probability further decreases. Simulation found that the extra amount of traffic due to a Bloom filter false positive is negligible.

Since newer FPGAs have multiple embedded microprocessors, transplanting the software into the embedded processor eliminates the need for external PCs.

6. CONCLUSION

As the rule set size of network intrusion detection and prevention systems grows and as network link speeds increase, the need for hardware accelerated network processing grows.

In this paper, a hybrid architecture is described that utilizes software's flexibility and hardware's high throughput. The hardware circuit acts as a pre-filter to reduce the traffic volume sent to the software. The hardware ensures that benign traffic passes through the system without the need for software processing. The reconfigurable hardware filter can be built in a stand-alone chassis or in a NIC mounted within a PC running the software NIDS. It is also possible to run the software in the embedded microprocessor of an FPGA.

The use of field programmable hardware enables a PC to effectively achieve an order of magnitude improvement in performance for the processing of Snort rules. Using current FPGA and PC technology, it is quite feasible to process 10 Gbps of data throughput using a single FPGA and one microprocessor.

By analyzing the output of hardware, most of the suspect packets are due to very few pre-filter rules, not real attacks. Thus, the number of packets sent to software is actually tens to hundreds of times greater than the number of harmful packets identified by software. This implies that there are still great opportunities to improve the filtering techniques. As future work, the hardware pre-filter can be enhanced further to achieve processing of data with even higher throughput and a more significant traffic reduction effect.

7. REFERENCES

[1] L. Schaelicke, T. Slabach, B. Moore, and C. Freeland, "Characterizing the Performance of Network Intrusion Detection Sensors," in *Proceedings of the Sixth International Symposium on Recent Advances in Intrusion Detection (RAID'03)*, 2003.

[2] W. Lee, J. B. Cabrera, A. Thomas, N. Balwalli, S. Saluja, and Y. Zhang, "Performance adaptation in real-time intrusion detection systems," in *Proceedings of the Fifth International Symposium on Recent Advances in Intrusion Detection (RAID 2002)*, ser. Lecture Notes in Computer Science. Zurich, Switzerland: Springer-Verlag, Oct. 2002.

[3] "Snort - The Open Source Network Intrusion Detection System," in *http://www.snort.org*.

[4] E. P. Markatos, S. Antonatos, M. Polychronakis, and K. G. Anagnostakis, "Exclusion-based signature matching for in-

trusion detection," in *IASTED International Conference on Communication and Computer Network (CCN'02)*, 2002.

[5] C. J. Coit, S. Staniford, and J. McAlerney, "Towards faster pattern matching for intrusion detection or exceeding the speed of snort," in *Proceedings of DISCEX II*, 2001.

[6] M. Fisk and G. Varghese, "Applying fast string matching to intrusion detection," 2004.

[7] N. Tuck, T. Sherwood, B. Calder, and G. Varghese, "Deterministic memory-efficient string matching algorithms for intrusion detection," in *Proceedings of IEEE Infocom*, 2004.

[8] 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*, 2002.

[9] K. Watanabe, N. Tsuruoka, and R. Himeno, "Performance of Network Intrusion Detection Cluster System," in *Proceedings of The 5th International Symposium on High Performance Computing (ISHPC-V)*, 2003.

[10] R. Sidhu and V. K. Prasanna, "Fast regualr expression matching using fpgas," in *IEEE Symposium on FCCM*, Apr. 2001.

[11] J. Moscola, J. Lockwood, R. P. Loui, and M. Pachos, "Implementation of a content-scanning module for an Internet firewall," in *IEEE Symposium on FCCM*, Apr. 2003.

[12] S. Dharmapurikar, P. Krishnamurthy, T. Sproull, and J. W. Lockwood, "Deep packet inspection using parallel Bloom filters," in *IEEE Symposium on High Performance Interconnects*, Aug. 2003.

[13] F. Yu, R. Katz, and T. Lakshman, "Gigabit Rate Packet Pattern-Matching Using TCAM," in *Proceedings of ICNP*, 2004.

[14] H. Song and J. Lockwood, "Efficient Packet Classification for Network Intrusion Detection Using FPGA," in *Proceedings of ACM/SIGDA 13th ACM International Symposium on Field-Programmable Gate Arrays (FPGA'05)*, 2005.

[15] 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 3rd Workshop on Network Processors and Applications (NP3)*, 2004.

[16] F. Braun, J. Lockwood, and M. Waldvogel, "Protocol wrappers for layered network packet processing in reconfigurable hardware," *IEEE Micro*, 2002.

[17] D. Schuehler and J. Lockwood, "A modular system for FPGA-based TCP flow processing in high-speed networks," in *14th International Conference on Field Programmable Logic and Applications (FPL)*, 2004.