Towards System-level Optimization for High Performance Unified Threat Management

Yaxuan Qi[1, 2], Baohua Yang[1, 3], Bo Xu[1, 3], Jun Li[1, 2]
[1]*Research Institute of Information Technology (RIIT), Tsinghua University, Beijing, China*
[2]*Tsinghua National Lab for Information Science and Technology, Beijing, China*
[3]*Department of Automation, Tsinghua University, Beijing, China*
yaxuan@tsinghua.edu.cn

Abstract

To build holistic protection against complex and blended network threats, multiple security features need to be integrated into a unified security architecture, which requires in Unified Threat Management (UTM). However, most existing UTMs operate by simply stringing together a number of security applications working independently without system level optimization that streamlines processing flow and leverages shared information and resources to reach high performance. In this paper, a generic framework is proposed to optimize the performance of UTMs at both algorithmic and architectural aspects by exploring the idea of Integrated Protocol Processing (IPP). The algorithm proposed in this paper improves overall protocol processing complexity of ACL and IDS from $\Theta(\log(M)+\log(N))$ *to* $\Theta(\log(M+N))$ *. Experiments on Intel IXP2850 network processor show that our scheme outperforms existing solutions with 30% increase of throughput.*

1. Introduction

Keeping network operations safe and efficient is more challenging than ever before, and network security has become one of the most critical issues facing today's Internet. Newly-emerging and ever-changing security issues come up with an alarming fact that: no organization can immune from security risk with traditional network security appliance. Because of the increased sophistication of security threats, standalone security products are not effective. To build holistic protection against complex and blended threats, multiple security features need to be integrated into unified security architecture, which results in Unified Threat Management (UTM).

UTM refers to a security appliance as a combination of hardware, software, and networking technologies whose primary function is to perform multiple security functions. According to IDC [1], the official definition of UTM is: "Products that include multiple security features integrated into one box. To be included in this category, an appliance must be able to perform network firewalling, network intrusion detection and prevention and gateway anti-virus. All of the capabilities in the appliance need not be used concurrently, but the functions must exist inherently in the appliance."

The value of UTM platforms is the simplicity and lower price given its "all-in-one" footprint. Some of the key benefits of UTMs include [2]:

- **Cost effectiveness**: By reducing the number of appliances, there is a lower up-front cost as well as lower management and support costs;
- **Easy to use**: Ideal for enterprises that lack the technical skills and resources to manage complex platforms;
- **Application level gateway**: The additional layer of security that a gateway device provides simply blocks network threats before they have the opportunity to enter the internal network.

While UTM solutions provide significant benefits, the design of many UTM appliances on the market today is a compromise of performance, functionality, price and simplicity. These compromises often include critical functionality such as firewalling and intrusion prevention, resulting in a product that delivers mediocre performance, restricted feature sets and limited scalability. Actually, the practical performance of a UTM appliance is typically just the performance of the firewall with the other security applications disabled or providing minimal functionality. Once these other security functions, such as IDS/IPS are enabled, performance can be reduced dramatically. This is because most of the existing UTMs operate by simply stringing together a number of security applications with little real consideration for the processing implications. These applications typically work independently of each other and do not leverage common information and resources. The lack of information interchange among different security applications results in:

- **Redundant packet classification:** Because security applications are commonly deployed in different independent Packet Processing Engines (PEs), while at the same time each application loads the multiple fields of packet header for

packet classification. Therefore, each packet header is loaded multiple times from the main memory and then processed by different PEs. This redundant packet header manipulation wastes the limited memory bandwidth and significantly hampers the overall performance.

- **Unnecessary deep inspection:** Deep inspection is very time-consuming and often the bottleneck of a UTM device. If deep inspection performed against the whole intrusion signature database were to be applied to every byte of network traffics, then multi-gigabit speed can be hardly achieved in practice. In fact, only malicious or dubious traffic needs to be processed by different security applications employing deep inspection functions before being allowed through, according security policies.

Thus, to reach high UTM performance, security applications should be integrated so as to effectively share common information and resources. In this paper, we propose a generic scheme to optimize the performance of UTMs at system level by exploring the idea of *Integrated Protocol Processing* (*IPP*). Main contributions of this paper are:

- **Integrated protocol processing algorithm:** Because unnecessary deep payload inspection can be significantly reduced by protocol analysis [3], we integrate the protocol analysis module of the time-consuming deep inspection application into the fast packet and session classification application to efficiently examine whether a packet need to take deep inspection. The algorithm proposed in this paper improves the complexity of ACL and IDS protocol processing from $\Theta(\log(M)+\log(N))$ to $\Theta(\log(M+N))$, where M and N are the size of ACL ruleset and IDS ruleset.
- **Network processor implementation:** Network Processors (NPs) have become core processing components for various Internet routers and switches because of their high programmability and optimized network processing capability. We implement the IPP algorithm on the Intel IXP2850 [4] network processor to evaluate the performance of a UTM prototype using integrated protocol processing. Experimental results show that our scheme significantly improves the performance of existing approaches with about 30% increase of throughput.

2. Background and Related Work

The protocol processing part of a network security application is responsible for the manipulations of networking protocols. Protocol processing involves several repeated basic operations, including packet classification, packet modification, session setup, session teardown, and statistics gathering. In this paper, we focus on the multidimensional packet classification operation, because this part of protocol processing is the key operation in our system-level integration of multiple security applications.

2.1. Protocol Processing Problem

Multidimensional packet classification for protocol processing classifies a packet into corresponding session according to the multiple fields of its header. Based on the F fields of the packet header, each rule specifies a *flow* that uniquely determines the action associated to the rule R. A packet P is said to match a particular rule R, if the i^{th} field of the header of P satisfies R[i], $\forall i$. If P matches multiple rules, the matching rule with the highest priority is returned. It has been proved that the best bounds for point location in N non-overlapping F-dimensional hyper-rectangles are $\Theta(N)$ storage with $\Theta(\log^{F-1} N)$ search time, or $\Theta(\log N)$ search time with $\Theta(N^F)$ storage in [5].

2.2. Protocol Processing Algorithm

Although the theoretical bounds make it impossible to design a single algorithm that performs well for all cases, fortunately, real-life rulesets have some inherent characteristics that can be exploited to reduce the complexity both in search time and memory space [6]. A variety of characteristics of real-life rulesets have been presented and exploited [7-9], and some best-known algorithms like HiCuts [10], HyperCuts [11], RFC [12], and HSM [13] achieve encouraging improvements in packet classification performance. Generally, these algorithms can be classified into two categories [6]:

- **Decision tree search algorithms:** HiCuts and HyperCuts belong to this category using decision trees for packet search. Generally, decision tree algorithms require less memory. However, the complexity of the decision trees often leads to implicit worst-case bounds and thus cannot ensure a stable worst-case classification speed.
- **Parallel search algorithms:** RFC and HSM perform independent parallel searches on indexed tables; the results of the table searches are combined in multiple phases to yield the final classification result. Algorithms using parallel search are very fast in term of classification speed while they require comparatively larger memory to store the crossproducting tables.

In this paper, we choose HSM to be the basis of the integrated protocol processing scheme, because HSM

algorithm provides both fast search speed and modest memory usage compared to other algorithms [13]. The $\Theta(\log N)$ search time of HSM provide explicit worst-case bound, and hence guarantees the protocol processing speed even with thousands of rules.

3. Integrated Protocol Processing

Protocol processing in UTMs involves repeated application of several basic operations. These mainly include packet classification for firewalling and protocol analysis for intrusion prevention. In order to get the maximum gain in overall performance, we need to design the protocol processing applications in a very efficient manner. However, because each processing engine need to manipulate multiple fields of packet header, they have to first load the packet header from the off-chip memory to local cache and then carry out modification and classification, and finally write the new header back into the external memory. Such read-modify-write operations are very time-consuming, and hence greatly impede the overall processing speed [14].

In this section, we propose an integrated protocol processing algorithm that efficiently combines the protocol processing functionalities of different security applications such as firewalling and IDS/IPS. Our algorithm handles multiple independent protocol processing rulesets by a compact data-structure using a HSM scheme.

3.1. Independent protocol processing

HSM performs parallel binary searches on multidimensional packet header, the result of the binary searches are combined in multiple *phases*. In the first phase, F fields of the packet header are **segmented** according to unique rule-projection intervals into multiple sub-spaces that are used to index into multiple memories (see Figure 1). Sub-spaces associated with the same rules are then **aggregated** and labeled with same sub-space ID (see Figure 2). In subsequent phases, earlier sub-space IDs obtained from one dimensional segmentation are recursively crossproducted with the sub-spaces obtained from other dimensions (see Figure 3). In the final phase, the memory yields the action. More detail of HSM can be found in [13].

3.2. Integrated Protocol Processing

Note that the most time-consuming part of HSM is the binary search because of its $\Theta(\log N)$ complexity. Thus, if we have two independent rulesets, each having M and N rules respectively, the original HSM algorithm requires two independent searches for each ruleset and hence has $\Theta(\log(M)+\log(N))$ temporal

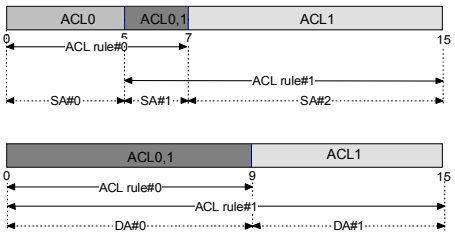

Figure 1. Space Segmentation. The two-dimensional (4-bit source address and 4-bit destination address) search space is segmented in each of the dimension according to two ACL rules. In each segment, there is a unique set of rules denoted by different SA# and DA#. SA and DA are segmentation tables of source and destination IP addresses, respectively.

AMT	SA#0	SA#1	SA#2
DA#0	1 (ACL0)	2(ACL1)	3(ACL1)
DA#1	0(N/A)	3(ACL1)	3(ACL1)

Figure 2. Space Aggregation. AMT is the address mapping table, which aggregates the sub-spaces denoted by the <SA#, DA#> pairs to a sub-space containing only the rule with the highest priority. SP and DP are segmentation tables of source and destination ports, respectively.

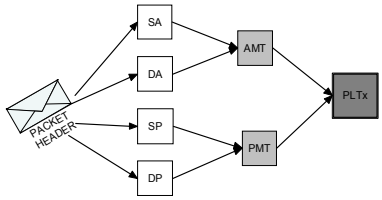

Figure 3. Hierarchical Space Mapping. AMT, PMT and PLT are hierarchical cross-producting tables for space aggregation. An incoming packet first gets indexes by looking up the four segmentation tables, and then using these indexes to trace the aggregation tables to yield finally classification result.

complexity. To improve the performance of binary search, the new algorithm should be able to handle multiple independent rulesets effectively. We propose to integrate multiple rulesets in the space segmentation step while remain independent tables for space aggregation. Specifically, there are two main steps:

- **Integrated space segmentation:** Different from the original HSM, space segmentation for each dimension is carried out not only according to ACL rules but also to the IDS rules. Because 16 bits can support up to 32K rules, we use 16-bit index for ACL classification and 16-bit for IDS protocol analysis to store the segmentation number. The number of segments then increases from $\Theta(M)$ to $\Theta(M+N)$, where M is the number of ACL rules, and N is the number of IDS rules. Because the binary search is now taken on

the integrated segmentations, the overall complexity becomes $\Theta(\log(M+N))$, which is much better than $\Theta(\log(M)+\log(N))$. Figure 4 illustrates how to implement the integrated segmentation scheme.

- **Independent space aggregation:** The space complexity of space aggregation tables are $\Theta(M^F)$ for M F-dimensional rules [13]. Thus, if we also integrate space aggregation tables, the overall space complexity will be $\Theta((M+N)^F)$. In practice, such an exponential increase of memory is not acceptable, so we retain the independency of each aggregation table and thus keeps the space complexity to be $\Theta(M^F+N^F)$. Because the number of memory accesses to these aggregation tables is small compared to the binary search on segmentation tables, the independent tables for multiple rulesets will not significantly impact the overall search time. Figure 5 illustrates the implement action of the independent space aggregation.

3.3. Summary of Integrated Protocol Processing

In real-life applications, it is not necessarily for all traffic to take the time-consuming content filtering, for intrusion detection/prevention or virus/spam scanning. For example, in Snort, most rules are set simply for HOME_NETS, i.e. most clients in HOME_NETS have the same level of security services. In contrast, IPP allows administrators to set more elaborate rules at network layer to have different level of security service for different hosts and servers in the internal networks. This potentially reduces a great amount of deep payload inspection workload while keeps high-level security for those important servers.

In addition, some security applications such as IDS and IPS require both range-match for protocol analysis and pattern-match for signature detection. This makes the data-structure of each application not consistent with each other and hence makes the corresponding implementations have larger code-size to perform more complicated operations. IPP solves this problem by separating the range-match codes from the pattern-match codes: on the one hand, the workload for packet header manipulation is reduced in each application; on the other hand, each packet header is loaded only once to reduce the number of memory accesses to off-chip memory, which is often the bottleneck of a UTM device.

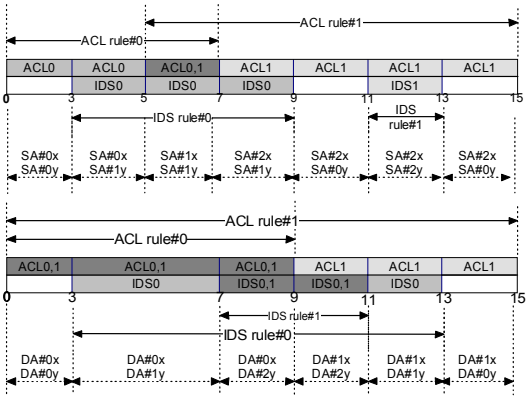

Figure 4. Integrated Space Segmentation. The two-dimensional (4-bit source address and 4-bit destination address) search space is segmented in each of the dimension according to 2 ACL rules and 2 IDS rules. In each segment, there is two independent unique set of rules. SA#x and DA#x denote the unique set of ACL rules, while SA#y and DA#y denote the unique set of IDS rules.

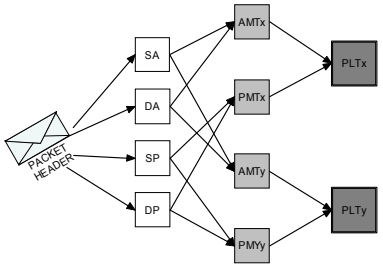

Figure 5. Independent Space Aggregation. Use #x indexes to traverse AMTx, PMTx and PLTx table to get the classification results for the ACL ruleset, which are commonly ACCEPT/DROP/EXEPTION. Use #y indexes to traverse AMTy, PMTy and PLTy table to get the finally IDS protocol analysis results, which may be a pointer to a specified ruleset for deep payload inspection.

4. Integrated Protocol Processing on Network Processor

Network processors are an emerging class of programmable processors used for implementation of packet processing applications in networking systems such as switches and routers. They are highly optimized for fast packet processing and I/O operations. NPs are typically characterized by distributed, multiprocessor, multithreaded architectures designed for hiding memory latencies in order to scale up to very high data rates. In this section, we provide a brief overview of the hardware architecture of Intel's IXP2850 Network Processor, and discuss how to implement IPP on this platform.

4.1. Intel IXP2850 NP Architecture

The architecture of IXP2850 NP is motivated by the need to provide a building block for multi-Gbps packet processing applications. A simplified block diagram of the Intel IXP2850 network processor is shown in

Figure 6 (curtsey of Intel SDK reference [15]). The major IXP2850 blocks are:

- **Intel XScale core:** general purpose 32-bit RISC processor used to initialize and manage the network processor, and can be used for higher layer network processing tasks.
- **Microengines:** 32-bit RISC processors optimized for network packet processing.
- **DRAM Controllers:** DRAM is used for data buffer storage.
- **SRAM Controllers:** SRAM is used for control information storage.
- **PCI Controller:** 64-bit PCI I/O bus. PCI can be used to connect to a Host processor or to attach PCI-compliant peripheral devices.
- **Media and Switch Fabric Interface:** interface for network framers and/or Switch Fabric that contains receive and transmit buffers.

4.2. Implementing Protocol Processing on IXP2850

The IXP2850 Network Processor receives Ethernet frames that carry IPv4 datagrams. The frames are assembled into packets and the Layer-2 (Ethernet) headers are removed. Then protocol processing applications are performed. Finally, packets are segmented into CSIX c-frames and transmitted to the CSIX fabric. The application design consists of the following Packet Processing Stages (PPSes) in a pipeline:

- **Packet receiving PPS:** The PPS reassembles m-packets one at a time to form the packet in DRAM. When a packet has been completely reassembled, it sends the packet to the next PPS over the communication pipe.
- **Protocol processing PPS:** This PPS includes main packet processing applications: a) Layer2 Ethernet header de-capsulate module; b) Protocol processing module, and c) IPv4 header validation and forwarding. Protocol processing algorithms are implemented in this PPS.
- **Packet scheduling PPS:** The PPS includes: a) Processes packet enqueue messages from the previous PPS; b) Sends cell dequeue messages to next PPS; c) Calculates the Weighted Random Early Detection drop probability for each queue.
- **Queue Manager:** The Queue Manager is set up to enqueue/dequeue packets.
- **CSIX transmitting PPS:** This PPS transmits the c-frame to the CSIX fabric.

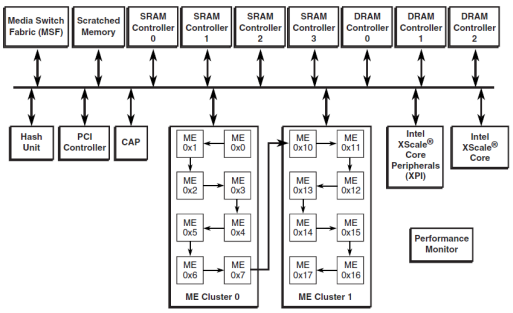

Figure 6. A simplified block diagram of the Intel IXP2850 network processor.

5. Performance Evaluation

To objectively evaluate the performance of the proposed IPP scheme, we focus on the integration of two common applications in UTMs: access control in firewalling and protocol analysis in IDS. We first evaluate the independent implementation of these two applications and then compare their performance with the integrated implementation.

5.1. Rulesets

Our study focuses on real-life rule sets. All the algorithms are evaluated on real-life firewall and IDS rules. Firewall rulesets are obtained from large enterprise networks and major ISPs, named as ACL01 ~ ACL07; IDS rules are extracted from Snort2.x and named as IDS01. All these rules are 5-dimensional with 32-bit source/destination IP addresses represented as prefixes, 16-bit source/destination port numbers represented as ranges, and 8-bit transport layer protocol. The size of ACL rulesets ranges from 68 to 1945, while the IDS01 ruleset contains 2835 rules. Each of the experiments is simulated with a pair of ACL and IDS rulesets. Because there is only one IDS ruleset (IDS01), we just list the ACL rulesets to denote different evaluation experiments. Network traffics are generated by the development workbench. Each traffic flow is an IPv4 TCP flow with 64-Byte Ethernet packets.

5.2. Evaluation Experiments

The most important metric of network processing algorithms is the worst-case memory access time, which indicates the temporal complexity. Table 1 shows the number of memory words accessed by the different two schemes in our comparison: FW+IDS refers to performing firewalling using ACL rules first, and then doing lookup against IDS rules; IPP refers to the proposed Integrated Protocol Processing, which performs ACL and IDS lookup simultaneously. Because one memory access in most existing operating systems and hardware platforms is word-oriented, we

use the total amount of 32bit-words as the memory access metric. From Table 1 we see that IPP scheme saves about 10 memory accesses, i.e. the worst-case search time of IPP is about 30% faster than that of the original FW+IDS scheme.

Table 2 shows the memory usage of each evaluation experiment. The memory usage is the total amount of memory required to store the lookup tables in Figure 3 and Figure 5. We see from Table 2 that the memory usage of IPP scheme is nearly the same as the original FW+IDS scheme. Therefore, the expense to trade memory for speed is very small. Actually, less than 1% extra memory is required.

5.3. Performance on Network Processor

To evaluate the hardware performance, we implement both schemes on Intel IXP2850 network processor. Microcode assembly was used as our programming language for its hardware compatibility and high efficiency. Packets receiving/transmitting and queuing/scheduling microblocks are implemented using Intel IXP2xxx SDK 4.0 building blocks. All these microblocks use 7 of 16 microengines to support a 10Gbps test bed. Our microcode is implemented in the protocol processing stage, using 6 microengines and 48 threads in parallel multiprocessing mode. Table 3 is the throughput measured for each pair of ACL and IDS rulesets. From this table, we see that the improvement of throughput is about 1Gbps by our IPP scheme, i.e. about 30% faster than FW+IDS schemes, which is in accordance with the worst-case memory accesses shown in Table 1.

6. Conclusion

To reach high UTM performance, security applications should be optimized at system-level rather than simply stringed together a number of security applications. In this paper, we propose an Integrated Protocol Processing scheme to resolve the key problems in existing UTMs: *Redundant Packet Classification* and *Unnecessary Deep Inspection*. The proposed algorithm improves the overall complexity of ACL and IDS protocol processing from $\Theta(\log(M)+\log(N))$ to $\Theta(\log(M+N))$. Analysis and experiments on Intel IXP2850 network processor show that our scheme outperforms existing algorithms with about 30% increase of throughput.

Table 1. Memory Access. (Unit: 32bit-word)

RULESET	#RULE	FW+IDS	IPP
ACL01	68	30	20
ACL02	136	31	22
ACL03	340	34	24
ACL04	500	34	24
ACL05	1,000	36	26
ACL06	1,530	36	26
ACL07	1,945	36	26

Table 2. Memory Usage. (Unit: Byte)

RULESET	#RULE	FW+IDS	IPP
ACL01	68	563,350	563,484
ACL02	136	584,680	584,944
ACL03	340	672,922	673,492
ACL04	500	609,880	610,664
ACL05	1,000	1,002,096	1,003,690
ACL06	1,530	898,422	899,902
ACL07	1,945	937,998	939,586

Table 3. Throughput. (Unit: Gigabits/Second)

RULESET	#RULE	FW+IDS	IPP
ACL01	68	3.72	4.64
ACL02	136	3.55	4.48
ACL03	340	3.37	4.46
ACL04	500	3.28	4.37
ACL05	1,000	3.10	4.03
ACL06	1,530	3.19	4.04
ACL07	1,945	3.16	3.97

7. References

[1] http://www.idc.com/

[2] http://www.itsecurity.com/

[3] H. Dreger, A. Feldmann, M. Mai, V. Paxson, and R. Sommer, "Dynamic Application-Layer Protocol Analysis for Network Intrusion Detection," Proc. of USENIX Security Symposium, 2006.

[4] http://www.intel.com/design/network/products/npfamily/

[5] M.H. Overmars and A.F. van der Stappen, "Range searching and point location among fat objects," Journal of Algorithms, 21(3), 1996.

[6] Y. Qi and J. Li, "Performance Evaluation and Improvement of Algorithmic Approaches for Packet Classification," Proc. of International Conference on Networking and Services, 2005.

[7] P. Gupta and N. McKewon, "Algorithms for Packet Classification," IEEE Network, March/April, 2001.

[8] D. E. Taylor, "Survey & Taxonomy of Packet Classification Techniques," Washington University in Saint-Louis, 2004.

[9] M. E. Kounavis, A. Kumar, H. Vin, R. Yavatkar, and A. T. Campbell, "Directions in Packet Classification for Network Processors," Proc. of Second Workshop on Network Processors (NP2), 2003.

[10] P. Gupta and N. McKeown, "Packet classification using hierarchical intelligent cuttings," Proc. of Hot Interconnects, 1999

[11] S. Singh, F. Baboescu, G. Varghese, and J. Wang, "Packet classification Using Multidimensional Cutting," Proc. of ACM SIGCOMM, 2003.

[12] P. Gupta and N. McKeown, "Packet classification on multiple fields," Proc. ACM SIGCOMM, 1999.

[13] B. Xu, D. Y. Jiang, and J. Li, "HSM: A Fast Packet Classification Algorithm", Proc. of the 19th Advanced Information Networking and Applications (AINA 2005), 2005.

[14] U. R. Naik and P. R. Chandra, "Designing High-performance Netwoking Applications," Intel Press, 2004.

[15] Intel IXP2850 Network Processor Hardware Reference Manual.

[16] www.snort.org/