2009 15th International Conference on Parallel and Distributed Systems

Towards High-Performance Network Intrusion Prevention System on Multi-core Network Services Processor

Xiang Wang[1], Yaxuan Qi[2], Baohua Yang[2], Yibo Xue[3,4] and Jun Li[3,4]
[1]*School of Software Engineering, University of Science and Technology of China, Hefei, China*
[2]*Dept. Automation, Tsinghua University, Beijing, China*
[3]*Research Institute of Information Technology, Tsinghua University, Beijing, China*
[4]*Tsinghua National Lab for Information Science and Technology, Beijing, China*
kojiroh@mail.ustc.edu.cn, ybh07@mails.tsinghua.edu.cn, {yaxuan, yiboxue, junl}@tsinghua.edu.cn

Abstract

Network intrusion prevention system (NIPS) becomes more complex due to the rapid growth of network bandwidth and requirement of network security. However existing solutions, either hardware-based or software-based cannot obtain a good tradeoff between performance and flexibility. In this paper, we propose a parallel NIPS architecture using emerging network services processor. To resolve the problems and bottlenecks of high-speed processing, we investigate the main design aspects which have dramatic impacts on most parallel network security system implementations: efficient and flexible pipeline and parallel processing, flow-level packet-order preserving, and latency hiding of deep packet inspection. To these key points, we address several optimizations and modifications with an architecture-aware design principle to guarantee high performance and flexibility of the NIPS on a network services processor implementation. Performance evaluation shows that, our prototype NIPS on Cavium OCTEON3860 processor can reach line-rate stateful inspection and multi-Gbps deep inspection performance.

1. Introduction

Facing the rapid growth of Internet bandwidth and continual emergence of new network applications, network systems require high-performance packet processing, which drives critical functions in network processing, such as policy processing and data flow control, to be merged into the data-plane [1]. And the functions in data-plane are becoming rich along with this trend. Take security territory for example, in order to perform finer-accuracy and higher-granularity processing, deep packet inspection (DPI) becomes the necessary module of security systems, such as NIPS and Anti-Virus Gateway. As all these new changes arise, it is a big challenge to the network system design and implementation. In general, network systems must match the following requirements: 1) network systems must have high performance adapted to the high-speed network; 2) network systems should be able to implement the ever-increasing applications; 3) network systems should be more extensible to be easily updated.

According to system requirements, device manufacturers have proposed different solutions. For high-end products, application specific integrated circuit (ASIC) / field programmable gate array (FPGA) based solution is commonly used [4][5]. This solution can achieve higher speed, but the disadvantages are also obvious: high risk, high cost and low flexibility. At low-end, the solution is based on X86 series generic processors and is able to meet the flexibility with low cost. However, this solution cannot be adapted to high-speed networks due to the limit of processor architecture. Thus, many chip manufacturers, such as Intel and Freescale [6][7], bring the concept of network processor (NP) with optimized architecture and dedicated instruction set to suffice line-rate packet processing. With the recent advance in technology, the industry extends the NP concept and proposes a newly designed multi-core network processor with security and application hardware acceleration engines, Network Services Processor (NSP) [3]. It not only reserves the traditional NP's advantages, but also provides high performance, high flexibility, low cost, and low power.

Deploying the NIPS, such a complicated network system, over NSP platform is also a challenge. The NIPS has complex data flow and different computing intensity tasks. It is required to balance the system's performance and flexibility. In this paper, we propose a parallel NIPS architecture based on Cavium OCTEON3860. Main contributions of this paper are:

1521-9097/09 $26.00 © 2009 IEEE
DOI 10.1109/ICPADS.2009.109

- **System Design:** We present an effective parallel NIPS architecture to meet both performance and flexibility. It also provides the facility of extending the system's functions. We describe all modules' functions in detail, and analyze the issues and bottlenecks. As a summary, architecture-aware NIPS design principles are proposed.
- **System Optimization:** To optimize the performance of NIPS on NSP platform, we use different coprocessors of NSP to break the following system bottlenecks: i) inflexibility of parallel and pipeline processing mode; ii) flow-level packet-order preserving over multi-cores; iii) slow pattern matching for deep packet inspection.
- **System Deployment:** The proposed NIPS is implemented as a prototype system on Cavium OCTEON3860 NSP. And experimental results show that the NIPS obtains line-rate stateful inspection throughput, and 2Gbps deep inspection performance.

The rest of this paper is organized as follows. Section 2 introduces the background of both NIPS and NSP. Section 3 presents the parallel NIPS architecture and discusses the challenges of implementation. Section 4 introduces the solutions to the bottlenecks in detail along with the guidance of network system design on multi-core NSP. Section 5 presents the experimental results and performance evaluation. Finally, section 6 concludes and discusses our work.

2. Background

Many research and industrial entities have proposed various solutions for high-performance NIPS. Most of them are implemented by adding stateful firewall and policy modules to the intrusion detection system (IDS) to make IDS obtain the ability of blocking attacking flow initiatively.

In general, there are hardware-based and software-based solutions. Among hardware-based implementations, A. Mitra, W. Najjar and L. Bhuyan compile the Perl compatible regular expressions (PCRE) library to FPGA to accelerate the intrusion detection of Snort [8], the most popular IDS which performs intrusion detection by comparing every incoming and outgoing packet against a rule set [11]. It achieves a significant speed up compared with the implementation on Intel architecture (IA), using a set of very limited rules. Sourdis and Pnevmatikatos implement independent comparison pipelines on FPGA to perform fast and large-scale pattern matching at a speed of multiple characters per clock cycle [9]. J. M. Gonzalez, V. Paxson, and N. Weaver propose a newly designed packet processing model – Shunting and implement it on FPGA to improve IPS performance [10]. They also adapt

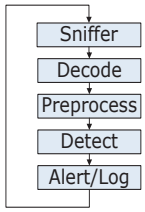

Figure 1. The packet processing loop of Snort

the Bro IDS to work with Shunting. The performance of these systems is well improved; however, it is obvious that all solutions based on hardware have high cost and low flexibility.

Among software-based implementations, most of the researches are around the optimization of Snort. Figure 1 shows the processing loop of Snort. Intel has proposed two kinds of Snort parallelism on IA [12], but only performing packets processing by multi-process of Snort or simply implementing a naive scheduler without the optimization for shared resources. These solutions cannot reach even 1Gbps throughput. Aiming at these unresolved issues, D.L. Schuff, Y. R. Choe and V. S. Pai research on data sharing between multi-thread in both packet-level and flow-level parallelization and achieve an average 1.09Gbps throughput [13]. Because this work is still based on the IA platform and just tested with local-stored traffic traces, the real performance on the advanced NSP platform is still unveiled.

As the solutions discussed above, the software-based IDS/IPS using IA platform can hardly meet the performance requirement of high-speed network, while hardware-based solutions are less flexible and often mean higher R&D costs. Therefore, it is necessary to search for other solutions to balance both sides.

The newly designed multi-core network processor with security and application hardware acceleration engines simplifies the network system design and implementation under current application background. It can be used in various networking equipments, including unified threat management appliances, content-aware switches, application-aware gateways, and storage networking equipments [2]. Its emergence provides us an appropriate approach to deploy the NIPS efficiently and flexibly.

Figure 2 shows the architecture of OCTEON3860 NSP. The processor has 16 MIPS cores and several coprocessors, including DFA hardware engine. The Packet Input Processing/Input Packet Data (PIP/IPD) unit receives packets from the wire, and then the Packet Order / Work (POW) unit schedules packets to different processing engines for packet processing. Finally packets are sent out from Packet Output Processing (PKO) unit. More details about OCTEON processor 3860 can be found in [14].

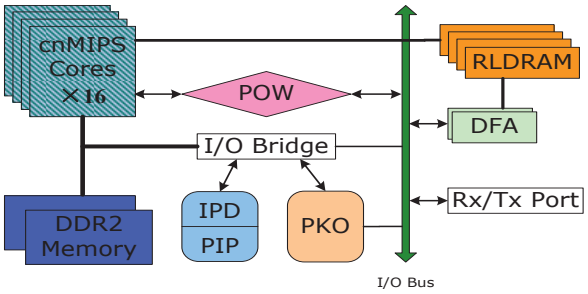

Figure 2. Cavium OCTEON main block diagram

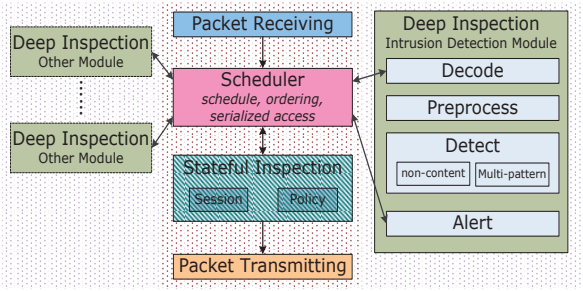

Figure 3. The parallel NIPS architecture

3. Parallel NIPS architecture

3.1. Design

The NIPS scans every incoming packet according to preset security policies. Once attacks in the traffic are detected, the NIPS takes prevention actions immediately. Due to different requirements, we divide the system into two different granularity inspections: stateful inspection and deep inspection. Stateful inspection tracks and validates connection-level packet information stored in packet headers, while deep inspection scans the overall packet payload for attack signatures. In consideration of both extensibility and flexibility, our NIPS architecture is proposed in figure 3:

Packet Receiving Module: It receives packets from wire, and performs packet analysis based on Layer 2 to Layer 4 headers. Then it passes the analysis results and packets to the scheduler module.

Scheduler Module: Based on the preprocessed results, this module schedules packets to different processing engines. It also provides packet-order preserving function and critical area protection for the traffic.

Stateful Inspection Module: It performs session management and stateful inspection. After the security policy is processed, it decides whether to pass packets to deep inspection module or only to send packets out.

Packet Transmitting Module: According to the priority, this module submits packets of different flows to different output queues, and then transmits packets to the wire.

The above four modules compose the fast-path to guarantee the rapid packet forwarding.

On the basis of the fast-path, the system is able to schedule packets to different upper processing engines by scheduler module. And the engine of upper processing can be extended according to the system requirement. For example, the anti-virus module can be easily added. In our prototype system, the upper processing module is only the intrusion detection module.

Intrusion Detection Module: This module is responsible for deep inspection – intrusion detection by matching the payload of packets to the preset rule sets. And it is separated into four submodules: Decoder, Preprocessor, Detection and Alert. Decoder performs protocol analysis from Layer 2 to Layer 4 for the following process. Preprocessor performs necessary examination and manipulation before packets are handed to the detection, including IP fragment and flow reassembly. Detection checks packets against various rule sets by examining the aspect or field of packets. Alert generates alert messages when attacks are detected.

3.2. Challenges

Processing Model: The NIPS has complex processing modules and different compute intensities of various granularity detections, therefore we map the inspections, stateful inspection and deep inspection, into two different data paths to guarantee the system's high forwarding efficiency. This not only avoids the compute-intensive function's long occupancy of computing resources, but also provides the flexibility of different combinations of pipeline processing in accordance with system requirements. Moreover, in order to improve the performance of each processing stage, we assign parallelizable tasks to different cores to make parallel processing. As a result, it is necessary to perform both pipeline processing and parallel processing in NIPS.

Packet-order Preserving: Being a network system, the NIPS need to send the packets in same flow by their incoming order. This is the principle of network system design and also is the requirement of high-layer protocols. Take transfer control protocol (TCP) for example, if the network system doesn't maintain the sequence, disorder will happen and be amplified during the transmitting. As a result, the receiver will get poor TCP performance. Besides, packets in the same flow must be kept in order when accessing the critical areas of processing loop, otherwise relative functions, such as stateful inspection will get error. Therefore, packet-order preserving scheme should also be carefully considered.

DPI Acceleration: DPI is the processing bottleneck, which is recognized by both academia and industry. Software-based algorithms are hardly adapted to high-speed traffic processing because: Firstly, during the DFA

graph walking, each node access of a DFA graph depends on the previous node and the next input character, and the performance is limited by the node access time. However, in software-based algorithm, DFA graphs are usually stored in memory whose access latency is large. Besides, DFA graph walking is character dependent and hence cannot get benefit from general CPU caching scheme. Also, large and/or multiple DFA graphs result in excessive capacity/conflict misses and unnecessary cache trashing [15]. So we need an optimized DFA processing engine and low latency memory used for storing the DFA walk graph.

Based on the discussion above, the problems are:

- How to build the NIPS architecture with efficient and flexible pipeline and parallel processing.
- How to guarantee the packet order at flow level with full use of system resources and low costs.
- How to accelerate detection processing in intrusion detection module.

4. Implementation of parallel NIPS on NSP

OCTEON processor includes a large variety of application specific offload and acceleration engines, such as compression/decompression, encryption/decryption, DFA and checksum engines. These engines help us to offload compute-intensive tasks of simple control logic and improve the system's performance.

4.1. All-in-one scheduler

As Section 3.2 shows, the system should perform efficient and flexible pipeline processing and parallel processing, and the scheduler module plays a great important role in the system. Since the generic processor is hardly competent for this task, the scheduler must be implemented in an efficient and specific coprocessor. In OCTEON processor, POW hardware unit is responsible for this function. This unit has the following important features [14]:

- Maintaining eight hardware queues for all incoming packets to provide different service levels. And the input work queues can be infinitely large if necessary.
- Scheduling and descheduling packets to different processing groups, which avoid consuming one core for this function. And the hardware supports a maximum of 16 groups
- Ordering and synchronization of packets. This unit associates a tag value/type tuple to each packet to support this feature.

In pipeline processing mode, the whole packet processing is separated into several stages, and each core handles one or more processing stages. Packets are passed from one stage to the next until the processing is completed. In OCTEON processor, we utilize the group scheme: each processing stage is mapped into one group; packets from different groups are processed by different processing engines. Besides this advantage, one processing engine can be set to receive packets from different groups, and call relative functions to process. The whole processing based on group scheme can be described as follows: when the processing of the task with group **X** is completed, the current processing engine changes the group number from **X** to **Y**, then deschedules the task and sends it back to POW. The POW picks the processing engine which is responsible for packets of group **Y** processing, and reschedules the task to it. Figure 4 shows the group-based pipeline process mapping:

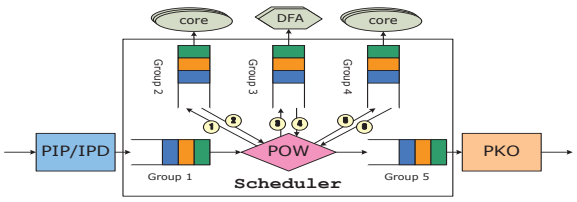

Figure 4. The group-based pipeline mapping

During each pipeline stage, we parallelize the processing to improve the performance. According to the packet's group and the status of the group-related cores, POW can choose the idle core to process the packet automatically.

In the implementation of our system, we set four groups for system processing: ORIG_GRP, TO_DO_ID _GRP, DFA_RESULT_GRP, and ID_RESULT_GRP which is separately used for representing the original packets, the packets which need intrusion detection after stateful inspection, the results of DFA processing, and the packets after intrusion detection. Six cores are selected for stateful inspection, and configured for processing packets from ORIG_GRP and ID_RESULT _GRP. The other ten cores are for intrusion detection along with the DFA hardware engine, and configured with TO_DO_ID_GRP and DFA_RESULT_GRP.

4.2. Flow-level packet-order preserving

As shown in Section 3.2, flow-level packet order must be guaranteed by every network system. From the network system's point of view, the packet-order preserving solution is typically software-based ordered thread execution [16], but it demands signal communication between every processing engine which is complicated for software design. POW's ORDERED tag type keeps the packets in inbound order, and simplifies the programming.

From the critical area's point of view, in multi-core environment, we can lock the critical area to make sure that only one packet in the same flow can access the critical area in a certain time. It is obvious that

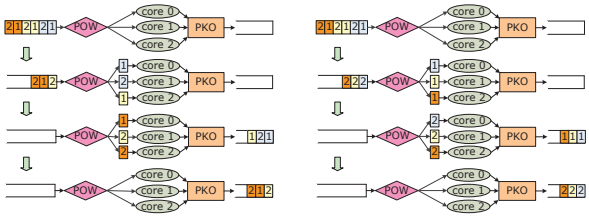

Figure 5. ORDERED tag type processing and ATOMIC tag type processing

lock/unlock is a waste of system resources. POW's ATOMIC tag type guarantees that the packets from the same flow, which means they have the same tag value, are serially processed in inbound order. So when accessing the critical areas, we need not to lock them, avoiding the consuming of the system resources. Figure 5 compares the ORDERED tag type packet processing with the ATOMIC tag type packet processing.

The tag scheme's advantage is not only keeping packets in order at different levels, but also switching flexibly between ORDERED and ATOMIC. When accessing the non-critical area, we can set the packets' tag value to ORDERED to parallelize the processing of the packets in the same flow; when accessing the critical area, we only perform a tag switch to ATOMIC, and the packets will be serially processed in ingress order automatically.

In the implementation of our prototype system, the session management and the TCP states transition in the stateful inspection module need the serial access, as well as the reassembly preprocess and the detection in the deep inspection module. So only before entering these critical areas, we perform a tag switch from ORDERED type to ATOMIC type. Thus, the other parts can be parallelized to make full use of the system resources.

4.3. DPI acceleration

In the intrusion detection module, the detection submodule occupies the most processing time, while the multi-pattern matching is the highest time-consuming part with an average of 48 percentage of the whole module processing [13]. Nowadays the software-based algorithm running on general CPU isn't suit for high-speed traffic processing, so we accelerate the multi-pattern matching by using DFA hardware engine. This acceleration unit has 16 DFA thread engines, the low-latency DRAM controller, and the instruction-input logic. In order to make further improvement, several optimizations have been done as follows:

Firstly, depending on the asynchronous mode of the DFA hardware engine, the generic processor performs necessary preprocess, including IP fragment and flow reassembly, then submits the packets to the DFA hardware engine for multi-pattern matching, and continues the other preprocesses and detections. The packets are shared between the generic processor and the DFA hardware engine. Once the DFA hardware engine completes the DFA walk, it passes the multi-pattern matching result to the POW. Then the generic processor gets the multi-pattern matching result from the POW and submits the final intrusion detection result to the stateful inspection module, including the results of other detections. In this way, we hide the latency of multi-pattern matching by parallelizing the multi-pattern match processing and other detections.

Secondly, as Section 3.2 discusses, during the DFA graph walk, each status transition is one memory access. The access latency of the SRAM which is the main memory of this system is above 40ns, so the memory access latency has great impact on system's performance. In the system implementation, we store the DFA walking graph in reduced latency DRAM (RLDRAM) whose access latency is about 15~20ns, and can be accessed through two dedicated low latency memory (LLM) channels.

4.4. System Deployment

Based on the discussion above, we finally implement the prototype system which is described as below:

The PIP/IPD unit performs preprocessing and checksum on every incoming packet and all packets here are labeled with ORIG_GRP and ORDERED tag type. According to the group type, the POW unit submits packets to the stateful inspection module. During the stateful inspection, a tag switch from ORDERED to ATOMIC will be executed after packets decapping, then the module will call session functions. After the stateful inspection, the traffic which needs deep inspection is labeled with TO_DO_ID_GRP and submitted back to POW unit. Then the POW assigns packets to the intrusion detection module. And in this module, multi-pattern matching will be offloaded to the DFA hardware engines, while the other detections will performed by generic processors. After the DFA walking results is received, the intrusion detection module will hand in these results accompanied with the other detection results to the stateful inspection module. The stateful inspection judges from the detection results to decide the action it will take.

In figure 6, we can state that all optimizations shown below speed up and flexiblize the system processing.

1) The efficient and specific coprocessor is used for scheduler, which guarantees the efficiency of scheduling. We can map different processing stages into different groups, which helps us to perform flexible pipeline and parallel processing.

2) Tag-based flow-level packet-order preserving scheme simplifies the programming, and its flexible tag switch helps us to perform both packet-level and flow-

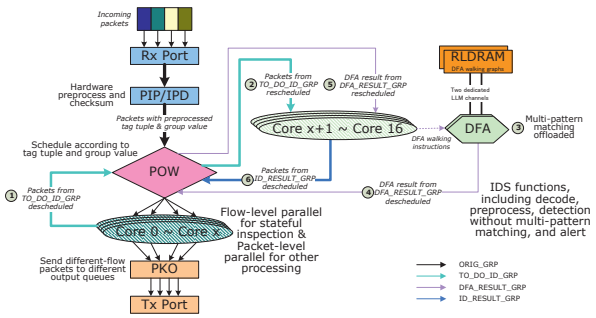

Figure 6. The NIPS packet processing flow

level parallel process during the whole packet processing with the full use of system's computing resources.

3) Hardware-based DPI processing improves the system performance. We modify the detection flow to parallelize the multi-pattern matching by DFA hardware engines and the other detections by generic processor, so we hide the latency of multi-pattern matching to a large extent. Besides this, RLDRAM with two LLM channels also reduce the memory access latency.

Based on the NIPS design and implementation, we provide several guidelines for building an efficient network system on NSP platform.

- Use group-based pipeline processing when necessary. Map different processing stages to different groups and call related functions. It is recommended to use different binary to do pipeline processing, if the performance must be guaranteed.
- Maximize the packet-level parallelism during each processing stage. Only switch to flow-level parallelism by tag switch when stateful function is called.
- It is recommended to use DFA hardware engine asynchronous mode to parallelize the generic processing and the DFA processing.

5. Experiments and performance analysis

In this section, the proposed parallel NIPS architecture is evaluated. First we will describe the development platform and test environments, then we show the comparing test results to illustrate our solution's advantage, and finally we present the prototype system's performance.

5.1. Development platform and test environments

The prototype system is running on Lanner MR950 series evaluation board. It includes one Cavium OCTEON3860 processor of 16 MIPS64 cores, eight RGMII interfaces connected to eight Gigabit ports which ensures maximum throughput and performance, two built-in DDR2 DIMM sockets with 2GB DDR2 400MHz main memory and 128MB RLDRAM onboard connected to two dedicated low latency memory channels to satisfy the requirements of the DPI applications. More detailed information can be found in [17].

The target system is based on Snort 2.4.3 which has around 5500 rules. We replace the multi-pattern matching algorithm with the DFA hardware walking to offload the compute intensive tasks, and several preprocessing sequence are modified for improving the performance. The Cavium Software Development Kit's version is 1.5.0 build 195.

There're two types of testing flow. One is the traffic generated by SmartBit 600, which is UDP flows of different packet size, and the other one is the traffic generated by two test machines which communicate with each other by multiple sessions using hyper text transfer protocol (HTTP) protocol.

5.2. Scheduler efficiency

Figure 7 shows the throughput of POW scheduler and generic MIPS64 core scheduler. The POW scheduler works under ATOMIC tag type, and the generic MIPS64 core scheduler only submits packets to the other 15 MIPS64 cores averagely without load balance and packet-order preserving functions. In order to investigate the tasks impacting on efficiency, we compare the pure scheduling and scheduling with jhash which is commonly used in Linux kernel network stack. We can see that regardless of the packet size, the POW scheduler can adapt to line-rate processing. On the contrary, the generic MIPS64 core scheduler is competent for only scheduling the packets of 256 Bytes or above to the other processing engines. And adding only the hash function significantly slows down the throughput. Therefore, it is wasteful of using one generic MIPS64 core to perform simple scheduling, and it is also susceptible to the amount of tasks. The scheduler should be implemented in a dedicated coprocessor.

5.3. Packet-order preserving

In classic NP, like Intel IXP28XX, the ordered thread execution is used for packet order preserving. An ordered critical section is used for reading the packets off the scratch ring form the previous stage. Then the engines process the packets, which may cause out of order during this stage. At the end of the dispatch loop, another ordered critical section is used for correcting the order. And inter-thread signaling is also used for the implementation [16]. From the discussion, we can see that the transmitting order should match the reading off order, and the first-in and last-out packet will cause the other processing

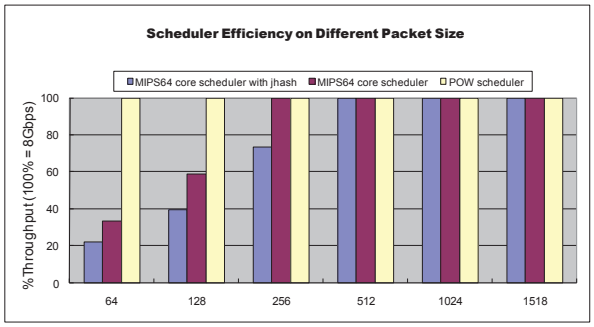

Figure 7. The scheduler efficiency on different packet size

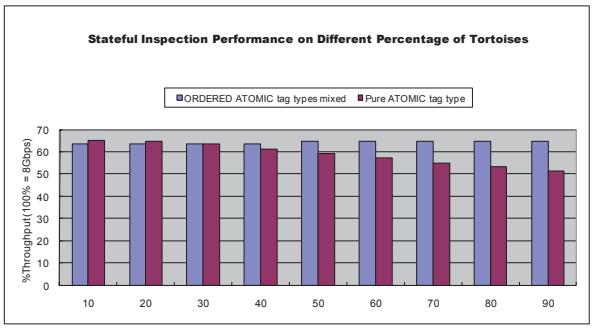

Figure 8. The stateful inspection performance on different percentage of tortoises

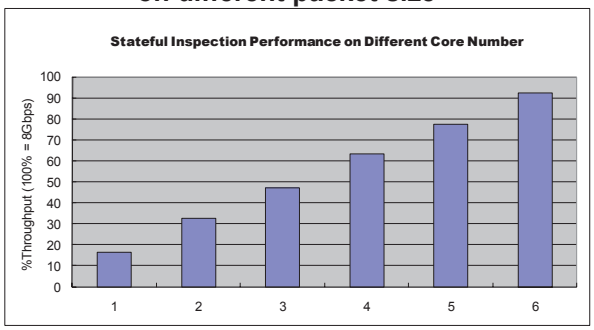

Figure 9. The stateful inspection performance on different core number

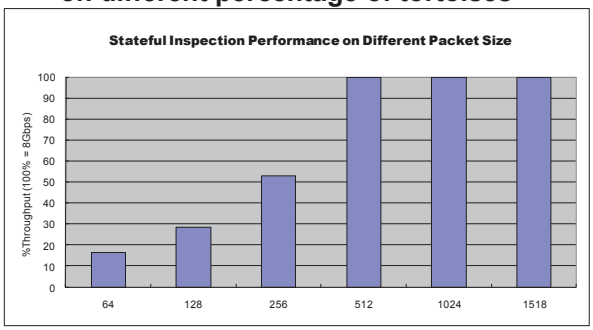

Figure 10. The stateful inspection performance on different packet size

engines waiting for order preserving. The ORDERED tag scheme ensures processing engines not idling away after processing packets, avoiding inter-thread signaling and leaving the order preserving to the POW scheduler. Therefore, it highly improves the system's performance and also is convenient for performing flexible pipeline and parallel processing discussed next.

5.4. Parallelization at different level

Figure 8 shows the comparison between pure ATOMIC tag type processing and ORDERED, ATOMIC tag types mixed processing. The throughput is tested on four MIPS64 cores. By Amdahl's Law, we can achieve higher speedup, if sequential parts of the program are fewer and the extra cost of parallelization is lower. In the case of fewer flows or high percentage of long lifetime flows, the pure ATOMIC tag type processing doesn't make full use of computing resources for its whole serialized processing. Therefore, we separate the whole processing into packet-level parallelization and flow-level parallelization using a tiny-cost tag switch between them. We can see that there is a slight slowdown of the throughput when mixed processing is used for multiple flows. While used for the real Internet traffic streams where the tortoises, defined as flows which have long lifetimes, and carries a large portion of the total bytes on links [18], the performance illustrated in figure 8 can be improved to a certain extent.

5.5. DFA walking accelerating

Table 1 shows the comparison of throughput between software DFA and hardware DFA using RLDRAM. The test is based on subset of Snort rules. We built HardDFA using DFA tools provided by Cavium, and built SoftDFA using AC [19] algorithms. The HardDFA is stored in RLDRAM, and the SoftDFA is stored in DDR2 DRAM. We can see that under the memory size limitation and whatever the DFA size is, the performance of hardware DFA is almost unchanged. On the contrary, the software DFA's performance goes down along with the increase of the DFA's size.

5.6. Stateful inspection performance

Figure 9 shows the performance of stateful inspection on different processing cores. When tested with the minimum Ethernet packets (64Bytes), stateful inspection with six cores can nearly reach the linear processing rate. Figure 10 shows the performance of stateful inspection on different packets size. When using one MIPS64 core, stateful inspection processing the 512-Byte packet can reach the linear processing rate.

5.7. Deep inspection performance

Table 2 shows the performance of deep inspection on different processing cores and different packet sizes. We

Table 1. The DFA performance on different DFA size

The Throughput (Gbps) of HardDFA and SoftDFA					
	320KB	855KB	3086KB	5675KB	9055KB
HardDFA	2.15	2.15	2.15	2.192	2.2
SoftDFA	1.12	1.12	0.87	0.524	0.426

Table 2. The deep inspection performance

The Deep Inspection performance: Throughput (Mbps)					
MTU	1 core	2 cores	4 cores	8 cores	10 cores
300	70	152	246	358	364
512	124	268	429	590	660
1024	245	543	852	1241	1548
1500	375	947	1539	2179	2185

can see that deep inspection using ten MIPS64 cores can reach 2.18Gbps throughput of normal Ethernet packets (MTU is set to 1500), which is limited by DFA hardware engine's bandwidth.

6. Conclusions and future works

In this paper, we present a parallel NIPS architecture and also implement the prototype system on NSP platform which is newly designed multi-core network processing platform. To resolve the problems and bottlenecks of high-speed processing, we make full use of the application specific offload and acceleration engines in the NSP. Experimental results show that, our prototype system can reach line-rate stateful inspection and multi-Gbps deep inspection performance.

Our proposed architecture shows high flexibility, and we can easily add more processing modules on this prototype system to enrich the system functions. Besides, our performance analysis indicates that the performance of deep inspection is limited by the DFA hardware engine's bandwidth. We can adjust the proportion of multi-pattern matching using DFA hardware engine to improve the system performance further. We will do more research on these two directions.

Acknowledgements

This work has been supported by the National High-Tech R&D Program (863 Program) of China under grant No.2007AA01Z468.

References

[1] Bob Wiest, "Evolution and requirements for DPI in network security infrastructure", in *the Asia-Pacific Advanced Network 25th Metting*, Jan. 2008.

[2] Cavium Networks Inc, "OCTEON™ MIPS64 Processors Architecture Advantages", Technical Report, June, 2007.

[3] Cavium Networks Inc, http://www.caviumnetworks.com/newsevents_octeon_montavista.html

[4] Juniper Networks Inc. "Juniper Networks ISG Series", Datasheet, 2004.

[5] Fortinet Inc. "Fortigate 5000 Series", Datasheet, 2006.

[6] Inetl Inc. "Intel Internet Exchange Architecture Network Processors: Flexible, Wire-Speed Processing from the Customer Premises to the Network Core White Paper", White Paper, 2002.

[7] Freescale Semiconductor Inc, "C-Port Network Processors", http://www.freescale.com/webapp/sps/site/homepage.jsp?nodeId=02VS0lDFTQ3126

[8] A. Mitra, W. Najjar, and L. Bhuyan, "Compiling PCRE to FPGA for Accelerating SNORT IDS", in *Proceedings of the 3rd ACM/IEEE Symposium on Architecture for networking and communications systems*, Dec, 2007.

[9] I. Sourdis and D. Pnevmatikatos, "Fast, Large-Scale String Match for a 10Gbps FPGA-based Network Intrusion Detection System", in *Proceedings of the 13th International Conference on Field Programmable Logic and Applications*, Sept, 2003.

[10] J. M. Gonzalez, V. Paxson, and N. Weaver, "Shunting: A Hardware/Software Architecture for Flexible, High-Performance Network Intrusion Prevention", in *Proceedings of the 14th ACM conference on Computer and communications security*, Oct, 2007.

[11] M.Roesch, "Snort — Lightweight Intrusion Detection for Networks", in *Proceedings of the 13th USENIX Conference on System Administration*, 1999.

[12] Intel Inc, "Supra-linear Packet Processing Performance with Intel Multi-core Processors", White Paper, 2006.

[13] D.L. Schuff, Y. R. Choe, and V. S. Pai, "Conservative vs. Optimistic Parallelization of Stateful Network Intrusion Detection", in *Proceedings of the 12th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2007.

[14] Cavium Networks Inc, "Cavium Networks OCTEON CN38XX Hardware Reference Manual", White Paper, Sept, 2008.

[15] Cavium Networks Inc, "OCTEON Content Processing", White Paper, Aug, 2005.

[16] U. R. Naik, and P. R. Chandra, *Designing High-Performance Networking Applications*, INTEL PRESS, Nov, 2004.

[17] Lanner Electronics Inc, "MR-950 User Manual", White Paper, 2007.

[18] Nevil Brownlee, and KC Claffy, "Understanding Internet Traffic Streams: Dragonflies and Tortoises", *IEEE Communications Magazine*, Oct, 2002.

[19] A. V. Aho and M. J. Corasick, Efficient string matching: an aid to bibliographic search. *Commun. ACM*, 18(6):333-340, 1975.