Asian Journal of Information Technology 13 (4): 235-241, 2014
ISSN: 1682-3915

A Technique to Parallelize Network Intrusion Detection using Multicore Network Processors

[1]M. Ravichandran and [2]C.S. Ravichandran
[1]Department of ECE, Saveetha School of Engineering, Saveetha University, Chennai, India
[2]Sri Ramakrishna Engineering College, Coimbatore, India

Abstract: As network speed increases tremendously than CPU and memory speed, it widens the gap between network and the end system. This poses a major challenge in the field of intrusion detection as the data has to be scanned at link speeds to detect the malicious packets. The prominent solution for this problem is addressed by the invent of multicore processors which cope up with increasing link speeds by offering parallelism and the required instruction sets to provide the necessary throughput. By making use of the multicore Network Processors it is possible to parallelize the intrusion detection activity at the data rate. In this scenario where processor is subject to computationally intensive task, performing stateful signature based analysis together with dynamic load balancing, without efficient parallelization is thorniest. Hence, an enhanced way of parallelization technique is proposed in this study to dynamically distribute the load among the core without jeopardizing the analysis reliability.

Key words: Intrusion Detection System, multicore processors, parallelization, dynamic load balancing, open MP, Network Processors

INTRODUCTION

An Intrusion Detection System (IDS) is broadly classified as Network Intrusion Detection System (NIDS) and Host Intrusion Detection System (HIDS). NIDS are located at strategic locations within the network to monitor to and fro traffic from all devices on the network. On the other hand, HIDS are run on individual host or devices on the network.

NIDS are a major security component in many network environments. They monitor network traffic for suspicious activity, providing alerts when they detect attacks. In real time deployments, merits of NIDS are questioned frequently as they raise frequent false alarms. NIDS face fundamental tradeoff between the detection rate on one hand and resource demand on the other. Choosing the tradeoff is an environment-specific decision. In particular, large high-performance environments (Gbps traffic networks) pose a major challenge for a NIDS. Due to the volume and heterogeneity of their traffic, short term characteristics are easily unpredictable. Hence, a NIDS needs to deal robustly with situations which are vastly different from any kind of average. Moreover, existing NIDS are not exploiting their full potential to mitigate the effect of the tradeoff they face. Even if a NIDS supports the installation of multiple detectors inside a network, these components work independently, only exchanging high level results such as alerts. At the same time, each detector maintains a great deal of internal state which remains hidden from the others. Yet combining all available knowledge promises to improve the detection rate without impacting either resource demands or false positive rate. The need to parallelize the intrusion detection with the help of multicore processors is due to the severity of the attacks. By keeping in mind the flexibility and inexpensive system costs its suitable to use multicore/multithreaded architectures rather than using extensive custom designed ASIC and execution models like FPGA which also provides multiple, concurrent processing.

The need for a dedicated Network Processor lies in the issue of scanning the network traffic at a line speed. This is accomplished by parallelizing the tasks of Intrusion detection in the Network Processor. The proposed system is framed in such a way that it takes full advantage of multicore/multithreaded architectures along with dynamic load balancing for network intrusion detection.

LITERATURE REVIEW

Sommer *et al.* (2009) framed an architecture customized for parallel execution of network attack

Corresponding Author: M. Ravichandran, Department of ECE, Saveetha School of Engineering, Saveetha University, Chennai, India

analysis. At its lowest layer, the architecture relies on an 'active network interface' that provides an in-line interface to the network, reading in packets and forwarding them only after they are fully inspected and deemed safe. The device dispatches packets to a set of threads that structure the processing as an event-based analysis model well suited to exploit many of the opportunities for concurrent execution (Sommer *et al*., 2009).

Dimopoulous *et al.* (2007) designed a memory efficient reconfigurable Aho-Corasick FSM algorithm. It has a desirable property that the processing time does not depend on the size or number of patterns in the network packets. It constructs a Deterministic Finite Automata (DFA) which is used to match all the patterns at once one byte at a time (Dimopoulos *et al*., 2007).

Hu (2006) suggested the idea of using Network Processors (NP) for intrusion detection by dividing the task into small pieces with each Micro Engine (ME) in NPs dealing with only one subset of detection operation. Packets received by the NPs are processed by the MEs by fixed pattern matching technique (Hu, 2006). Colajanni and Marchetti (2006) proposed a parallel NIDS architecture that is able to provide us with fully reliable analysis, high performance and scalability. The load balancing mechanism proposed in this study aims at distributing the traffic among a configurable number of parallel sensors, so that each sensor is approached by manageable amount of network traffic (Colajanni and Marchetti, 2006).

Pangrazio (2008) has suggested that the rules could be partitioned into blocks for the cores and pushed out to the device. Here, one core is used to control the interface and the rule set is broken up into fifteen blocks. The packets would then be passed from cores to cores sequentially. Since, packets only trigger an alert once, the first rule set should contain the most common or simplest rules.

This would push the more difficult rules to the latter cores where they would see less traffic and allow the string or pattern matching to have more time. This would reduce overall load to the IXP and increase the throughput. Breaking the rules into subsets allow parallel processing similar to that of the superscalar processors that have been proposed for high throughput applications (Pangrazio, 2008).

Bos and Huang (2004) aimed at detecting worms at high speeds by matching the payload of network packets against worm signatures at the lowest possible levels of the processing hierarchy (microengines of an IXP1200 network processor). The solution employs the Aho-Corasick algorithm in a parallel fashion where each microengine processes a subset of network traffic. To allow for large patterns as well as a large number of rules, the signatures are stored in off-chip memory. Using an old version of the IXP Network Processors (the IXP1200), the system is capable of handling close to 200 Mbps with full content scan for realistic threats (Bos and Huang, 2004).

EXISTING SYSTEM

The existing architecture for parallel network intrusion detection is customized for parallel execution of network attack analysis. The objective is to build a highly parallel, inline network intrusion prevention systems that can fully exploit the power of modern and future commodity hardware.

The architecture encompasses a front end Active Network Interface (ANI) and backend analysis component. ANI is implemented at the lowest layer of architecture and it reads the packet from the interface. It makes the routing decision based on the information available in the connection and host table, if no such information is available corresponding to the flow identified then packet is passed to analysis component which works as a back end.

Structure of the existing architecture: The ANI drives its despatch decisions based on a large connection table indexed by packet header five-tuple. The table yields a routing decision for each packet either thread which will analyse the packet, ANI should drop the packet directly without further processing or ANI should forward the packet directly (to enable some forms of off-loading). There is an analogous table indexed by IP addresses to provide per-host blocking and also default routing for packets not found in either table (Sommer *et al*., 2009) (Fig. 1).

Active network interfaces: The ANI is a stateful device whose functionality can be dynamically refined by the backend analysis engine show in Fig. 1. The ANI is responsible for: routing copies of packets to the appropriate analysis threads, retaining packets until signalled by the analysis engine to either forward or drop them and supporting alteration of packet content.

Thread-aware routing: The first task of the parallel analysis pipeline is flow demultiplexing; routing packets to analysis threads. This task is assigned to the ANI. For each packet, it first decides which thread (s) is in-charge of the corresponding flow. The ANI then appends a copy of the packet to the packet queue of the core running that thread. The ANI determines to which thread the packet has to be routed.

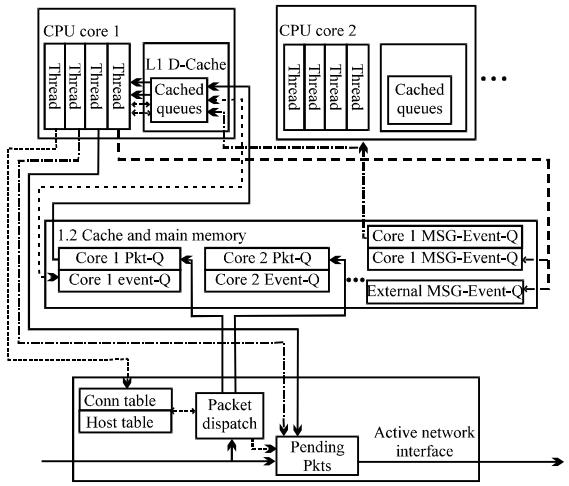

Fig. 1: Existing architecture for parallel execution of network attack analysis (Sommer *et al.*, 2009)

Selective packet forwarding: The ANI is an in-line element that for a given packet it either forwards it or drops it. The ANI itself does not decide to forward or drop the packet until it is decided by the backend. To avoid requiring the backend to transmit entire packets back to the ANI for forwarding when the ANI routes a packet to a thread, it includes a packet descriptor that the backend can subsequently use to refer to the particular packet.

Analysis component: The analysis proceeds in stages as depicted in study. The initial stages concern low-level tasks such as TCP stream reassembly and normalization, suitable to a single thread of execution. This stage requires very little inter-thread communication. It outputs events parameterized with parsed packet headers (since, normalization already requires header analysis) and payload byte streams (for TCP).

The next stage performs application-layer protocol parsing. This stage can significantly benefit from parallelizable execution. The outputs from this stage are events reflecting application level control information (requests and responses) with associated protocol data units. Finally, these events are consumed by multiple high-level analyzers that detect attacks both within application dialogs and across multiple connections and hosts.

A crucial point is that to extract parallelism at each stage of the pipeline to gain the maximal performance gain. In Fig. 1, vertical boxes reflect different types of analyses, progressing in semantic level and breadth from left to right. The progression of arrows indicates how information flows from one level to the next with the thickness of an arrow indicating the relative volume of data within the flow. Thinner arrows indicate fewer threads of analysis that need to execute at the next stage relative to the previous stage hence if the latter stage offers less opportunities for parallel execution but also will be presented with fewer flows to analyze then researchers can still keep the pipeline full as they analyze flows at increasingly high levels. Of particular note is the large degree of task-level parallelism which can easily be leveraged by multi-core and multi-threaded processors.

Limitations of existing system: The limitations of the existing system are listed:

- Generic processor cannot provide satisfactory performance due to overheads on handling system interrupts, moving packets through PCI bus and ISA not optimized for networking applications
- Scalability of components in active network interface like connection table and host table is limited hence it can not cope up with the increased traffic from multiple hosts
- Operational deployment of generic processor is not extended to data communication line rate due to its load sharing by other applications
- This architecture does not provide dynamic load balancing, i.e., once he packets are scheduled to one thread they are not reassigned

Solutions to the problems in the existing system are addressed in the proposed research.

PROPOSED SYSTEM

Network Intrusion Detection System (NIDS) are becoming a worthful element in a modern network infrastructure for guaranteeing the security of complex information systems. A NIDS is used to inspect the network traffic with the aim of looking for evidences of illicit activities and malicious network packets. To control all the traffic flowing through a network, a NIDS has to perform a stateful analysis on each packet. This necessitates a NIDS to track and reassemble each distinct connection.

The proposed system is designed in such a way that it can operate at line speed of the network to identify the intrusion with the help of Network Processors. An innovative parallel NID architecture that achieves high performance by combining cores parallel with no necessity for hardware components is proposed. The proposed NIDS can effectively scale and deal with increasing traffic and an innovative load balancing

technique that dynamically dispatches incoming traffic to the analysis engines. The architecture allows the NIDS to inspect high speed links with no packet loss and no negative impact on the accuracy of the traffic analysis that remain reliable and stateful (Colajanni and Marchetti, 2006).

Solutions to the limitations of the existing system: To compensate the performance degradation experienced through generic processor a device is specifically designed called Network Processor (NP) which is used for packet processing at high speeds by sharing the workload between a number of independent RISC processors.

Cores present in the Network Processors allow the flexibility to program according to the need. Hence, the scalability issues faced in the existing system due to the use of hardware component can be overcome with the help of these programmable cores.

In order to make the Network Processor to operate at link speed it is directly linked with the number of packets processed per second. As the number of packets processing per second increases then more number of packets can be dispatched thereby achieving the packet reception at link speed.

A dynamic load balancing mechanism is introduced which dynamically reassign the packets to corresponding cores for analysis according to the rules specified. These rules were altered according to the arrival pattern of packets.

Network application categorization: Network applications normally entail into three major tasks: data path, control path and management path tasks which are so categorized according to their functionality. Network applications are executed in the data path, the control path and the management path of a Network Processor. Data path tasks which must run at line rates and require high performance, execute core functionalities such as receiving/transmitting from/to MAC devices, packet forwarding classification and queuing and scheduling. For example, forwarding packets from the input to output ports must be executed at line rates to avoid dropping of packets. If there are no packet dependencies among the packets being processed, multiple packets can be processed in parallel to achieve line rates.

Control path tasks which are less time-critical execute such control functionalities such as table maintenance, routing, signalling and policy management. ATM virtual circuit set up and tear down is the example of control path task. Control path tasks present little data parallelism and help Network Processor to execute data path tasks. System initialization/configuration and management protocols belong to the management path tasks. It should also be noted that the management path tasks present very little data parallelism.

Features of network applications: Network applications have many special features compared to general applications. These features such as the presence of network protocol layers, their data intensive and branch intensive character, the existence of significant packet parallelism and the need to detect packet dependencies are presented.

Network protocol layers: Most networks are organized as a series of layers to reduce their design complexity. The layered model of network protocols is proposed by the International Standards Organization (ISO) and the model is called the ISO OSI (Open Systems Interconnection) Reference Model. The layered structure produces parallelism to deal with packet concurrently.

Packet parallelism: The packet is the base unit of work for network applications. Packets may be independent and may be processed concurrently. There are several types of parallelism that could be exploited in Network Processors in addition to the more commonly encountered Instruction Level Parallelism (ILP) and Thread Level Parallelism (TLP); they are packet level parallelism and intra-packet parallelism.

Packet level parallelism: If each incoming packet is independent of the other packets being processed, the incoming packets can be concurrently processed in the several independent processing units of a Network Processor. This kind of parallelism is called packet level parallelism.

Intra-packet parallelism: During the processing of each packet if the tasks to process a packet are independent, then those tasks can be executed in parallel. For example source MAC address manipulation and destination MAC address manipulation can be executed at the same time in layer 2. This kind of parallelism is called intra-packet parallelism.

Packet dependency: When many packets are being processed simultaneously in a multiprocessor or in a multithreaded environment, packet dependencies between two packets may or may not exist. If there exist packet dependence then it requires synchronization to obtain correct processing results. When packets are processed with static rules such as in the case of stateless firewall and forwarding engines there are no packet dependencies

because the code working in each packet does not need to modify the rules. Therefore, synchronization is not required since packets are processed independently.

If packets are from the same TCP connection and it is necessary to update the state in memory for the purpose of encryption or TCP state maintenance between the processing of two packets, a packet dependency exists. Packet dependency also occurs when updating traffic management counters and routing or address translation tables. If a packet dependency exists, some sort of synchronization is required.

Parallized network processor architecture

Multithreaded architectures: Multithreaded architectures have been developed to tolerate memory access latencies. When an instruction from one thread stalls due to a long-latency operation, the thread is made inactive and no more instruction of the thread are fetched from the instruction cache while instruction from other threads are placed into the execute pipelines. As a result a multithreaded architecture allows for multiple threads to share the functional units of a single processor. It addresses the drawbacks of superscalar machines which offer more performance with added functional units but suffer from a low utilization of those functional units.

Fine-grained multithreading: Fine grained multithreading which switches threads and issues instructions from only one thread at each clock cycle. The advantage of fine-grained multithreading is in the hiding of latency caused by both short and long stalls.

The disadvantage of fine-grained multithreading is that it slows down the execution of instructions which are ready to be executed, since instructions which are ready to be executed are delayed by instructions from other threads. Since, only one thread can issue instructions at each clock cycle in fine-grained multithreading there are no fully empty issue slots but a significant number of idle issue slots within individual clock cycles still exist.

Simultaneous multithreading: Simultaneous Multithreading (SMT) can issue multiple instructions from multiple threads in a single cycle and schedule them dynamically so as to concurrently exploit Thread-Level Parallelism (TLP) and Instruction Level Parallelism (ILP). With register renaming and dynamic scheduling, multiple instructions from independent threads can be issued without any dependencies among them. SMT enhances performance across a wide range of applications without significant hardware cost or major architectural changes.

Chip multi processor: Chip Multi Processors (CMP) is actually multiprocessors which have a full set of architectural resources on the same die. A CMP exploits TLP by executing different threads in parallel on different processors while SMT exploits TLP by issuing instructions from different threads with large issue widths within a single processor. A CMP consists of a single thread processor cores which are relatively simpler than general purpose processors where multiple threads are executed in parallel across multiple single-thread processor cores. If an application cannot be effectively decomposed into threads, the single-thread processor cores are not fully utilized. In worst case, when an application can not be decomposed into threads, only one single-thread processor core can be assigned to the task.

PROPOSED ARCHITECTURE

Network Processors play a crucial role in packet processing such as packet forwarding in the network equipment. In the Network Processors of the first generation, general purpose processors were used. Network applications were mostly software based and new features can be easily added. However, scalability was severely limited and some these processors even failed to meet the speed requirements. As a result, ASIC-based Network Processors were introduced as second generation processors. ASIC-based Network Processors are typically used to forward traffic at very high rates. Their disadvantages include high development costs, long time to market and little flexibility. As internet traffic continues increasing rapidly and the protocols are becoming more dynamic and sophisticated, Network Processors are required to be very flexible as well as fast. Thus, Network Processors that have an instruction set specialized for network applications, flexibility and support speed of line rates have emerged as the third generation Network Processors. In this study, a parallel NIDS architecture is proposed. It is a complex infrastructure which needs proper configuration. Depending on the size of incoming traffic, the packets are sliced as per the rules set. A well designed set of slicing rules has to satisfy the following main properties:

- Packets belonging to the same connection have to be routed towards the same cores. This is necessary to trace and reassemble all connections as required by the stateful traffic analysis characterizing the proposed architecture
- Network traffic should be equally distributed among the available cores. This allows the architecture to achieve good load balancing properties

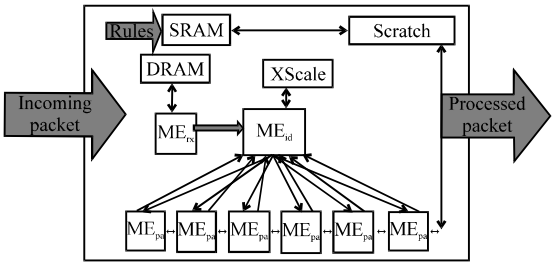

Fig. 2: Parallelized Network Processor architecture for intrusion detection

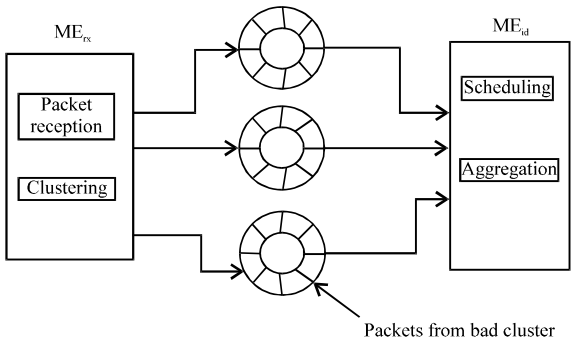

Fig. 3: Packet flow between the Micro Engines (ME) in a Network Processor

- Both the above requirements can be satisfied through a well designed set of slicing rules. While it is rather simple to write slicing rules that preserve transport level connections, achieving a reasonable load balancing among cores is a non-trivial task

Load balancing: Another innovation of the proposed architecture is represented by its ability of dynamically migrating the states of analyzed connections between different cores.

This feature can be used to achieve load balancing among Micro Engines (ME) thus increasing the scalability and the overall effectiveness of the parallelized architecture (Fig. 2 and 3).

Packet processing: The traffic source feeds a core (reception) that it captures every Ethernet frame and sends the frame to one of the directly connected cores (dispatcher). The core (reception) is the only centralized element and has to manage traffic at the same throughput of the network link. Architecture scalability is limited by the number of cores (dispatcher) that the core (reception) is able to handle. In order to keep the computational cost

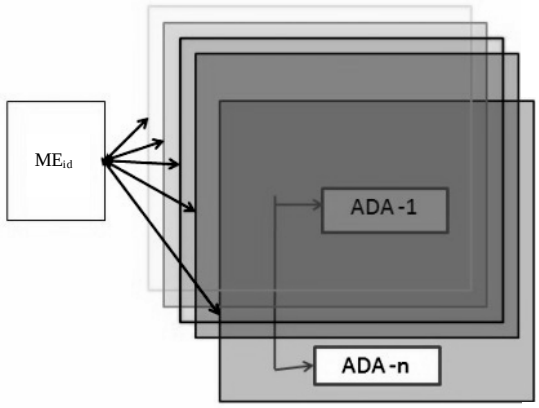

Fig. 4: Parallelization of Anomaly Detection Algorithm (ADA) for detecting intrusion

reasonably low, scattering operations must be as simple as possible. The cores of Network Processor capture every frame from the input interface and have to determine the destination cores of each of them. That operation can not be executed by the cores because in the implementation the destination cores are selected on the basis of a frame analysis that is quite complex operation.

Indeed, each core implements a set of slicing rules that are used to classify every received Ethernet frame. In particular, the implementation of the cores (dispatcher) makes it possible to select the destination sensor on the basis of various features such as protocols, IP addresses and port numbers. Slicing rules need to be carefully designed, so as to route every frame towards the Cores (analyzer) that may need it to detect an attack. This is one of the most important tasks since we need to perform a stateful traffic analysis. Hence, researchers must guarantee that every frame belonging to the same transport level connection is routed to the same Cores (analyzer).

Network Processors (NP) provide affordable processing power to support the real time requirements of network security applications. The real challenge lies in programming the NP which tender continued growth in CPU performance. To achieve greater usability of NPs in IDS, a new technique called parallel vector processing which utilizes micro engines of Network Processors for anomaly detection is proposed. This technique makes use of data mining approaches which brings software parallelization in NP based on Asymmetric Multi-Processing (AMP). The objective of optimal utilization of NPs is targeted by keeping relevance between time and data thereby achieving overall speedup (Fig. 4).

CONCLUSION

In this study, a parallelization technique for anomaly based Intrusion Detection algorithm is proposed. The parallelized algorithm can be executed by different cores of the Network Processor simultaneously thereby increasing the overall throughput.

REFERENCES

Bos, H. and K. Huang, 2004. A network intrusion detection system on IXP1200 Network Processors with support for large rule sets. Technical Report 2004-02. http://www.cs.vu.nl/~herbertb/papers/trixpid.pdf.

Colajanni, M. and M. Marchetti, 2006. A parallel architecture for stateful intrusion detection in high traffic networks. Proceedings of the IEEE/IST Workshop on Monitoring, Attack Detection and Mitigation, September 28-29, 2006, Tuebingen, Germany, pp: 1-7.

Dimopoulos, V., I. Papaefstathiou and D. Pnevmatikatos, 2007. A memory-efficient reconfigurable Aho-Corasick FSM implementation for intrusion detection systems. Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation, July 16-19, 2007, Samos, pp: 186-193.

Hu, Y., 2006. Embedded Network Processor based Parallel Intrusion Detection. In: Embedded Systems: Modeling, Technology and Applications, Hommel, G. and S. Huanye (Eds.). Springer, The Netherlands, ISBN-13: 9781402049330, pp: 93-100.

Pangrazio, G., 2008. Intel IXP Network Processor based intrusion detection. October 16, 2008, SANS Institute, USA. http://www.sans.org/reading-room/white papers/detection/intel-ixp-network-processor-based-intrusion-detection-32919.

Sommer, R., V. Paxson and N. Weaver, 2009. An architecture for exploiting multi-core processors to parallelize network intrusion prevention. Concurrency Computat.: Pract. Exp., 219: 1255-1279.