AntiWorm NPU-based Parallel Bloom Filters for TCP/IP Content Processing in Giga-Ethernet LAN

Zhen Chen, Chuang Lin, senior member, IEEE, Jia Ni, Dong-Hua Ruan, Bo Zheng, Yi-Xin Jiang,

Xue-Hai Peng, Yang Wang, An-an Luo, Bing Zhu, Yao Yue, Feng-Yuan Ren

***Abstract*—TCP/IP protocol suite carries most application data in Internet. TCP flow retrieval has more security meanings than the IP packet payload. Hence, monitoring the TCP flow has more strength than only monitoring the IP packet payload in the AntiWorm system. The main idea of this paper is to use the flexibility and high performance of Network Processors to scan TCP flow for locating worm's binary codes, and cut off their propagation. A stateful TCP flow inspection engine is implemented based on IXP Network Processor, which can monitor about 512K flows. The performance issues about IXP Network Processors are evaluated and collected, and an analysis is made for further optimizing the system performance. The system is also demonstrated and proved by using the Internet traces and real assaults of Worms. Software Package TCPScanner 1.0 is also given as a software release of the research.**

***Index Terms*—Network Security, Worms, Network Processors, TCP/IP Protocol suite, Parallel Bloom Filter, Deep Packet Inspection, Stateful TCP inspection.**

I. Introduction

A. TCP/IP/Ethernet

Ethernet is a popular packet-switched LAN technology invented at Xerox PARC in the early 1970s. Xerox Corporation, Intel Corporation, and Digital Equipment Corporation standardized the Ethernet in 1978. Gigabit Ethernet builds on top of the Ethernet protocol, but increases speed tenfold over Fast Ethernet to 1 gigabit per second (Gbps). Gigabit Ethernet was standardized in June 1998, and is widely used in high-speed LAN and MAN for connectivity.

TCP/IP is probably the most useful protocol suite in Internet. It forms the base technology for a global internet that connects Hosts, Routers and Switches. Statistics show that the TCP/IP traffic accounts for 86.5% in the total Internet traffic, UDP/IP traffic accounts for 12.8%, and the miscellaneous 0.7%. TCP/IP guarantees the reliable, in-order delivery of byte stream. TCP supports a flow of byte streams in each direction and also includes a flow-control mechanism for each direction.

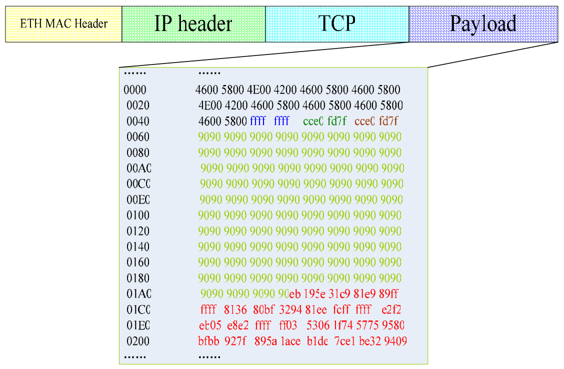
Many security issues are also concerned with the TCP/IP, such as Browser security [10], Worms[11], Bots[12], Spyware, Spam mails, etc. Hence, it is important to audit the TCP traffic to identify the possible threat for protection purpose.

B. Worms

A Worm is a kind of malicious code, which can be self-reproduced and propagates over a network. The life cycle of a worm is as follows: (Phase 1) Target discovery: the worm in the infected host tries to locate a victim host; (Phase 2) Carrier: First, make the new host prepares to receive the worm code. Then, transfer the worm code (in plain text (unencrypted) mode or in encryption way) to the new host; (Phase 3) Activation: change the new host's registry, and make it to reference the worm code. Then the process repeats.

The following symptoms of active worms can be used for the detection of Worms.

In the victim Target discovery Stage, a simple strategy the worm used is scanning, including sequential and random scanning, e.g. CodeRed. Scanning is a highly anomalous behavior, so it can be effectively detected by devices.



```
......          ......
0000            4600 5800 4E00 4200 4600 5800 4600 5800
0020            4E00 4200 4600 5800 4600 5800 4600 5800
0040            4600 5800 ffff  ffff   cce0 fd7f  cce0 fd7f
0060            9090 9090 9090 9090 9090 9090 9090 9090
0080            9090 9090 9090 9090 9090 9090 9090 9090
00AC            9090 9090 9090 9090 9090 9090 9090 9090
00CC            9090 9090 9090 9090 9090 9090 9090 9090
00EC            9090 9090 9090 9090 9090 9090 9090 9090
0100            9090 9090 9090 9090 9090 9090 9090 9090
0120            9090 9090 9090 9090 9090 9090 9090 9090
0140            9090 9090 9090 9090 9090 9090 9090 9090
0160            9090 9090 9090 9090 9090 9090 9090 9090
0180            9090 9090 9090 9090 9090 9090 9090 9090
01AC            9090 9090 9090 90eb 195e 31c9 81e9 89ff
01CC            ffff  8136  80bf  3294 81ee  fcff  ffff   e2f2
01EC            eb05  e8e2  ffff   ff03   5306 1f74 5775 9580
0200            bfbb  927f   895a 1ace  b1de  7ce1 be32 9409
......          ......
```

Figure 1. Blaster's executable carried in TCP payload during propagation

In the Carrier stage, a worm can either actively transfer itself from host to host, or it can be carried along as part of normal communication, e.g. Worm Blaster, a.k.a. RPC DCOM worm, exploits the victim's RPC (Remote Procedure Call) flaw and make the victim connect back to the infecting machine using TFTP (Trivial File Transmission Protocol) to download the worm code. Figure 1 shows the payload of a packet of RPC

This work is funded by IXA University Program of Intel Research Council (No. 0411A66) and partially supported by Intel China Research Council and Postdoctoral Science Foundation of China (No.023130017). This work is also supported by the National Grand Fundamental Research 973 Program of China (No. 2003CB314804), the National Natural Science Foundation of China (No. 90412012， 60429202, 60273009, 60372019, 60432030 and 90104002), NSFC and RGC (No. 60218003).

The authors are with the Institute of Network technology, Department of Computer Science and Technology, Tsinghua University, Beijing, China.

Corresponding author: Zhen Chen, Phone: 86-10-62772487, Fax: 86-10-62771138, E-mail: zhenchen@csnet1.cs.tsinghua.edu.cn.

DCOM worm v2.2, where the WORM executable is illustrated in details.

Some AntiVirus Tools, such as ClamAV[13] also use segments of virus's executable code as a signature to identify Worms.

C. Intel IXP2400 Network Processors

The Network Processor Unit (NPU) [2-8] enables network researchers and developers to import the latest network services while maintaining high throughput and low latency. In this paper, Intel IXP Network Processors IXP2400 is used and will be introduced as follows [14-21].

IXP2400 integrates the following function blocks: 8 Multi-Threaded MicroEngines (2 Clusters), XScale Core, 4K 32-bit word Scratchpad Memory, 2 QDR SRAM Interface controllers, 1 DDR DRAM Interface controller, Hardware Hash Unit, Media and Switch Fabric(MSF) Interface, Half Duplex OC-48/2.5 Gbps Ethernet Interface, and other Peripherals Components.

A ME consists of the following resources: executable data path (ALU), 16-entry CAM, CRC unit, 4K × 40-bit instruction MicroStore, local CSR, CMD FICO, 640 32-bit Local Memory, 256 GPRs (Bank A and Bank B), 128 SRAM Read (Write) Transfer registers, 128 SRAM Read (Write) Transfer registers, Eight threads per ME (no overhead for context switching), 128 Next Neighbor registers etc.

Software frameworks can be used in programming the Network Processor. This programming focuses primarily on writing data plane components utilizing MEv2 MicroEngines. Logical networking functions named MicroBlocks are used in the fast path processing. Each MicroBlock is a macro (Assembly Language) or MicroEngine C function written using underlying low-level libraries.

The paper is structured as follows: Section 2 introduces the design of AntiWorm TCP content processing System using Parallel Bloom filters. Section 3 shows the implementation of the system in detail. In Section 4, experiments are carried out for demonstrating the working of AntiWorm system and the experiment results are shown. Section 5 introduces the software package of the research. Section 6 presents the further work. Finally, Section 7 concludes the whole paper.

II. System Design

A. The Principle

The main idea of this paper is to use the flexibility and high performance of Network Processors to detect and scan TCP flow to locate worm codes, and cut off their propagation. In this paper, an AntiWorm system based on TCP/IP Content Processing is implemented based on IXP2400. Parallel Bloom Filters are implemented in TCP/IP content processing engine, but other signatures based search functions can also be integrated. The TCP/IP content processing engine is stateful, and can realize some more complex Intrusion Detection Functions.

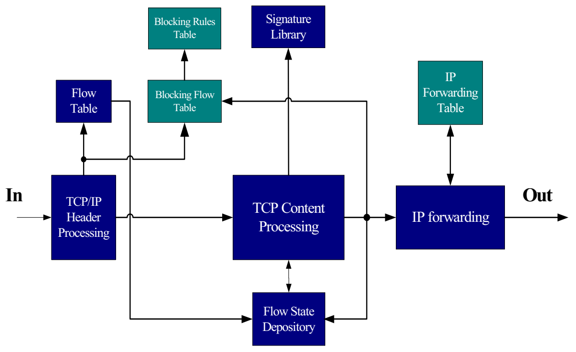


Figure 2. Logical Function Blocks in TCP/IP Content Processing System

The system is deployed in one side (ingress or egress) of network, but both sides can also both be deployed and cooperated for a more effective purpose. The logical function blocks in the system are shown in Figure 2, while the physical function block in IXP2400 are shown in Figure 3.

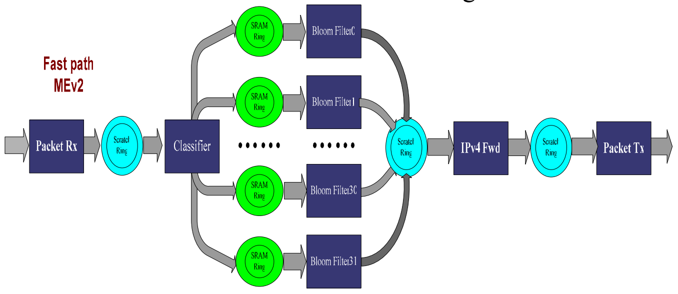


Figure 3. Function Blocks of TCP flow Content Processing System

B. The Flow Procedure of a packet

A brief introduction of the journey of a packet will be helpful in understanding the system.

A TCP/IP packet arrives from one physical port. It is received by RX Module, and then passed to the Classifier Module complying with some Blocking Rules. (Note the blocking function is optional, can be bypassed directly).

Classifier Module looks up all entries in flow state table and decide whether to add the flow state into Flow table or not. The flow state table is organized in hash table, and the collisions are resolved by using linked-list. If the TCP/IP packet belongs to a new flow, then a new entry will be created and added into the flow state table. If not, the flow state of the TCP flow which the packet belongs to will be retrieved and sent to the corresponding SRAM ring along with the buffer handle of the packet etc. The flow state and application data are kept in a record called Flow Descriptor. Flow Descriptors are stored in DRAM for economical reason.

An optional function of Classifier Module is Traffic Monitor. It collects information such as traffic volume of a certain source IP, or a certain destination port. An unusual large flow from a non-server host or to an unknown port is supposed to be a sign of worm activity. To achieve a tradeoff between monitor accuracy and performance, a multistage filters technique proposed by Cristian Estan and George Varghese [30, 31] is used to account the 1% largest flows. The amount of ICMP destination unreachable message is also counted, while a large amount possibly means an active random-scanning worm [41].

With Traffic Monitor enabled, we could scan TCP context only when there is a sign of worm outbreak, so relief the context processing engines.

TCP content processing engines drain the packets from the respective SRAM ring and perform Bloom Filters action. If suspicious, the packet is exactly compared with the signatures one by one. An optional function is draining the specified TCP flow into XScale buffer, where a ClamAV agent is responsible for scanning the total flow the TCP for some Worms. After handling a packet, the Flow Descriptor in DRAM is updated.

Then it is turn for IPv4 Forwarder Module, which maintains the route table and forwards the packet into transmitting queue. Finally the TX Module transmits the packet out of the system.

C. Hash Table Index

A Hash table is used to index memory that stores each flow's state. Gracefully handling hash table collisions is difficult for real-time systems. To ensure proper monitoring of all flows, the state store manager can chain a linked list of flow state records off of the appropriate hash entry.

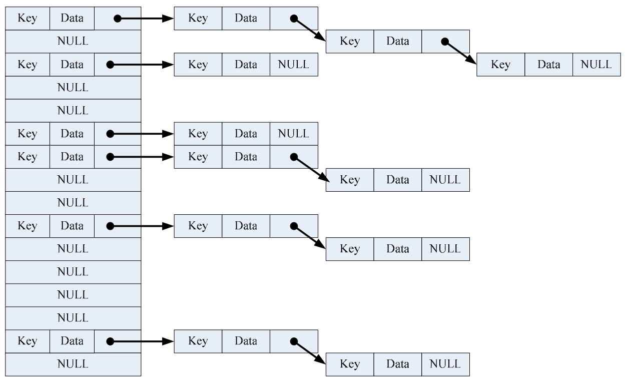


Figure 4. Hash Table Structure

D. Bloom Filters

To quick identify worms' signatures in TCP byte stream flow, Bloom Filter technique [22-25] is used in TCP/IP Content Processing Engine. Countable Bloom Filters is implemented in practical system for scalable reasons.

Definition 1. (**Bloom Filter**). A Bloom Filter is used to represent a set $S=\{s_1,s_2,\cdots,s_n\}$ of n elements from a universe U . It consists of an array of m bits, initially all set to 0. A Bloom Filter uses k independent random hash functions $h_1,\cdots,h_k$ with range $\{0,\cdots,m-1\}$. These hash functions map each element $s \in S$, the bit at adderss $h_i(s)$ is set to 1 for $1 \le i \le k$. A Bloom Filter may yield a false positive, where it suggests that an element x is in S even though it is not, but it will not generate a false negative event.

Given the assumption that hash functions are perfectly random, let $p=e^{-kn/m}$, the false positive probability f is given

$$f=\left(1-\left(1-\frac{1}{m}\right)^{kn}\right)^k \approx \left(1-e^{-kn/m}\right)^k=(1-p)^k$$

(Eq. 1)

Definition 2. (**Counting Bloom Filter**). A Counting Bloom Filter is a Bloom Filter enhanced with a vector of counters corresponding to each bit in the bit vector (BV) for scalability reason. Whenever a counting Bloom Filter adds or deletes a member, it increments or decrements the counters corresponding bit in the bit vector. Only when a counter changes from one to zero, it clears the corresponding bit in BV. Therefore, these counters change only during the addition and deletion of strings in the Bloom Filter.

E. Flow state based content scanning

The Bloom Filters based TCP content scanning engines processes TCP byte streams from the TCP Classifier Module. The content scanning engines can match a signature that spans across multiple packets in the same flow. When a packet of a certain flow reaches the content scanning engine, the content scanning engine must restore the Flow Descriptor for that flow before starting the matching operation on that packet. When it has finished processing the packet, the content scanning engine must save the flow's new Flow Descriptor.

Packets are transferred to the content scanning engine along with a data structure shown in **Table 1**. The content-scanning engine firstly loads the flow state of that flow after receiving a packet from the corresponding SRAM rings. If the content scanning engine finds no matches in the packet, then the packet can pass through the module. If a match is discovered, then it notifies the flow blocking module to block the flow, terminates the connection, or lets the data pass through. It can also send out alert messages in response to content matches to XScale.

Each content-scanning engine processes data one byte at a time. Having four MEs (32 threads) for content-scanning engines in parallel and processing 32 flows concurrently, as Figure 3 shows. The system dispatches incoming packets to one of the content scanning engines based on a hash of the flow ID by the TCP Classifier Module. Dispatching packets in this way can eliminate the possibility of hazards that two content-scanning engines were simultaneously processing packets from the same flow.

III. IMPLEMENTATIONS

A. System Requirements

TCP Flow State Stores:

(1) The amount of TCP flows monitored is about 512K (2^{29});

(2) Flow state store is 64Bytes each in size;

(3) The packets in TCP/IP stream are received in order. Beside, the packets which are retransmitted are ignored.

Bloom Filters:

(1) The size of Signature Library is about 128;

(2) The false-positive probability of the Bloom Filter should be less than 2^{-8};

(3) The size of signature is fixed and 16 bytes long.

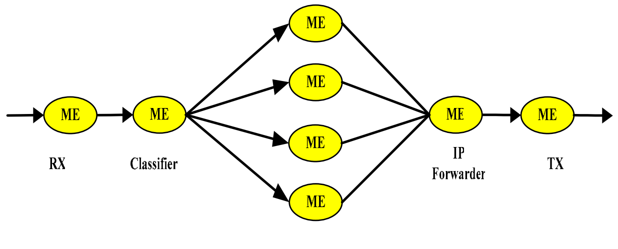


Figure 5. Two Practical Implementation of Parallel Bloom Filters

B. ME Allocation

In current design, as an integrated system, RX Module takes one ME, a Classifier Module occupies one ME, IP Forwarder Module occupies one ME, and TX Module takes one ME, the remaining 4 MEs are used as TCP Flow Content Processing Engines. See Figure 5.

C. Flow State Table

A resequencing container (AISR, Asynchronous Insert Synchronous Removing) is needed to place in front end of the system. Hence the assumption is asserted that the packet is coming in order. The flow state table is organized in the form of hash table, see Figure 4. A new flow state store is created and added into hash table in the following two cases: one is the arrival of the TCP SYN packet, the other is the arriving packet can not match any flow state entry in existence. The old flow state store is also flushed in the following two cases: one is the reception of the TCP FIN packet, the other is a flow state is too old, which is indexed in the timestamp marked in the flow state. The XScale is responsible for clearing the idle TCP Flow Descriptor at a fixed interval. A process is woken up in the specified interval, frees the idle flow state store according to the timestamp marked in the last flow state update and current timestamp.

In a common-case scenario in which there's no more than one entry per hash bucket, each packet requires a total of one read and one write operations to the DRAM: a 16-word read to retrieve flow state, and a 16-word write to update flow states.

Table 1 illustrates the TCP Flow State Record (Flow Descriptor), which reflects the design considerations aforementioned.

D. Data Structure

1) Data Structure in Scratchpad Ring

Scratchpad ring is used to store the intermediate results of Packet handling in context pipeline.

2) Data Structure in Creating SRAM Ring

SRAM controller provides a queue-array element to access a queue or a ring. When used as a ring, a SRAM ring is very similar to a Scratchpad ring, in a way that the data being stored in Scratchpad memory, the data is stored in a contiguous block of SRAM. Also, SRAM rings can be configured into sizes of 512, 1K, 2K, 4K, …, 64K long words to support a larger FIFO. When used to implement a ring, a queue-array element contains the head and tail pointers, the size of the ring, and the current number of elements (count) on the ring.

3) Data Structure in SRAM Queue

SRAM controllers for the Intel IXP2400 and IXP2800 network Processors support a data structure called Q-array, which provides hardware-supported basic queue management. For applications requiring a few small FIFOs, Scratchpad rings are sufficient. However, Scratchpad rings are not sufficient for applications requiring more than 16 FIFOs. In such case, the IXP2XXX processor's solution is to use SRAM-based FIFOs. The IXP2XXX hardware can support as many FIFOs as can fit within SRAM memory and provides access to these FIFOs through a 64-element cache (per SRAM controller).

4) Flow State Records (or Flow Descriptor)

Table 1 shows the most important data structure in the system, Flow State Record for a given flow. The total size is 64Bytes which a ME can retrieves all data in a single memory operation. The Classifier Module maintains one of these records for each flow that the content scanning engine need to reference.

Of the 64 bytes of data stored for each flow, 32 bytes is used to maintain flow state. The additional 32 bytes of state store for each flow can hold the application-specific data for each flow context.

E. Software Framework

1) Dispatch Loop

The dispatch loop combines MicroBlocks on a MicroEngine and implements the data flow between them. The dispatch loop also caches commonly used variables in registers or local memory. These variables can be accessed by MicroBlocks using a set of macros or microC functions. The dispatch loop also provides source and sink blocks to send and receive packets to the Intel XScale core and to send packets to a different MicroBlock group.

Table 1. The TCP Flow State Record (or Flow Descriptor) (LSB-MSB)

LW	Bits	Size	Field	Description
0	31:0	32	Hash Value	The hash value of five-tuple
1	31:0	32	Next Flow Pointer	Address of the next Flow state in DRAM
2	31:0	32	Source IP address	Source IP address
3	31:0	32	Destination IP address	Destination IP address
4	31:0	32	TCP Source Port	Source port
5	31:0	32	TCP Dest Port	Destination port
6	31:0	32	Protocol	Protocol(TCP, UDP etc.)
7	31:0	32	Sequence Number	TCP packet sequence number
8	31:0	32	Payload offset	Source and Destination Port
9	31:0	32	CODE BITS	Some Flags for Packet handling
10	31:0	32	Time Stamp	The time mark for the arrival packet(update)
11	31:0	32	Remaining length	Index the last remaining byte length
12	31:0	32	1st word	The first word of the last packet in flow
13	31:0	32	2nd word	The second word of the last packet in flow
14	31:0	32	3rd word	The third word of the last packet in flow
15	31:0	32	4th word	The fourth word of the last packet in flow

A dispatch loop is specific to the application being targeted. The intent is for the mircoblocks to be as reusable as possible. The dispatch loop and the internal implementation of its associated helper macros or functions may be optimized for a specific application.

2) RX Microblock

RX MicroBlocks interface with the MSF and reassemble incoming m-packets (RBUFS) into complete packets for further processing by down stream packet processing code. For each packet, the packet data is written to DRAM, the packet descriptor (meta-data) is written to SRAM and a handle to the packet is written to a Scratchpad ring for use by the packet processing stage.

3) Classifier Module

Classifier Module is assigned to one ME for scalable reason. The minimal function of classifier module is described as follows: **(Phase 0)** read the packet information (buf_handle) from Scratchpad ring enqueued by the upstream RX Module; **(Phase 1)** extract the five-tuple of the TCP/IP flow packet; **(Phase 2**) hash the extracted five-tuple of the TCP flow packet; **(Phase 3)** search the entries (or maybe create a new entry) in the TCP Flow State Table, retrieve the Flow Descriptor's address; **(Phase 4)** write the packet handle information into the working SRAM ring, dispatch the incoming TCP packet to a SRAM ring.

The Classifier computes the hash value of five-tuple and initiates a Flow Descriptor retrieval operation by reading flow state store. The Classifier dispatches incoming packets to one of the TCP processing engines based on the hash value. Dispatching packets in this way eliminates the possibility of hazards that two content-scanning engines are simultaneously processing packets from the same flow.

For the purpose of load balancing and packet order, the Classifier ME operates in the Thread Ordered Model and the Hash table searching is marked as critical section for mutual exclusion. The SRAM ring usage is for load balancing in the stricter packet order maintenance. See Figure 6 for more details.

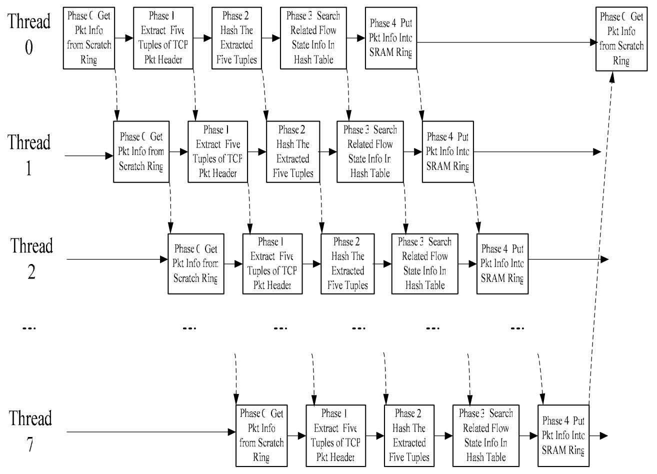

Thread 0
Thread 1
Thread 2
Thread 7
Phase 0 Get Pkt Info from Scratch Ring
Phase 1 Extract Five Tuples of TCP Pkt Header
Phase 2 Hash The Extracted Five Tuples
Phase 3 Search Related Flow State Info In Hash Table
Phase 4 Put Pkt Info Into SRAM Ring

Figure 6. Thread Ordered Model Implemented In Packet Classifier Module

4) *TCP content processing (Bloom Filter Module)*

Currently, the content processing engine based on Bloom Filter is implemented. The main function of the Bloom Filter Module is described as follows: read the TCP packet info (buf_handle) and TCP Flow Descriptor pointer, scan the packet in the joint consideration of the state stores of the last packet and whole of the current proceeding packet.

The aim of TCP content-processing is to recover TCP byte stream and scan them by Bloom Filter. Therefore, if there are two consecutive packets of same stream, the second one can not enter into the Bloom Filter until the first one passes through the Bloom Filter entirely. Otherwise, it can not merge the data of two packets and a mistake occurs.

Thus, it is incorrect to simply put all the packets into one unique ring in order. That may cause two consecutive packets of the same stream are fetched by two paralleled Bloom Filter threads concurrently, which is prohibited.

In order to avoid such troubles and guarantee all the packets in the same flow are transferred to one thread, a separate ring is assigned to each thread running Bloom Filter Module. The hash value of the five-tuple is used to choose the ring where the packet is transferred.

The Bloom Filter thread fetches packets of TCP stream from corresponding ring in order. If the packet payload length is less than the length of signatures (i.e. 16 bytes), it means that the payload can not hold one signature. The Local Memory is used to cache the combination of the previous packet's tail of the same TCP flow with current packet's payload. If the combined data's length is less than 16 bytes, then the data are simply written back to Flow Descriptor which can be used by next packet, none operations are applied in this case. Otherwise, the combined data should be scanned by Bloom Filter to check whether signature is concealed and then be written back to Flow Descriptor. If a signature is broken into 2 parts contained in the two packets, the Bloom Filter will alarm in such condition. If the packet's payload is longer than 16 bytes, normal scanning action is taken. At last the packet's tail data are written into Flow Descriptor.

In any step mentioned above, if any matches, the corresponding TCP flow should be handed over to upper layer processing. One possible solution is that the TCP flow byte stream is drained and transferred to XScale core. The AntiWorm agent of ClamAV running in XScale takes charge of inspecting the flow carefully. Another operation includes simple drop the packet from the TCP flow.

5) *TX Module*

The Packet Transmit block runs on a single MicroEngine and receives transmitting requests from the queue manager. Since the transmit requests are packet based, the MicroBlock needs to segment the payload into mpackets, copy them into TBUFs and validate them for the MSF to transmit. Each port has its own Scratchpad queue to store the packet enqueued.

IV. EXPERIMENTS

A. *Simulation Environment*

1) *Features*

Two independent QDR SRAM controllers, each controller can address up to 64MB.

(1) Two independent QDR SRAM controllers, each 4MB, total 8M.

(2) 64MB DDR SDRAM.

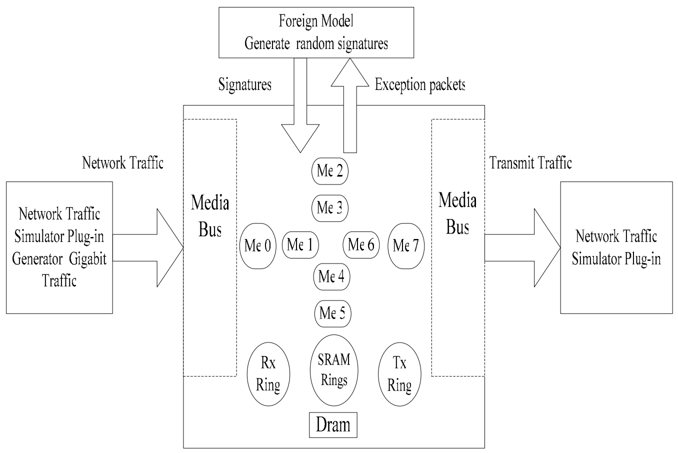


Figure 7. The simulation environments.

The simulation experiments are carried out on Intel Developer WorkBench V4.1, which is a cycle and data accurate simulator. In this experiment, we use both *Foreign Model* and *Network Traffic Simulation Plug-in* on WorkBench, see Figure 7. The Foreign Model is used to generate 128 signatures and writes them to the Local Memory of the MicroEngines with Bloom Filter Module. The Network Traffic Simulation Plug-in is used to simulate a Gigabit Packet Generator, which generate packets and send them to the media bus of IXP 2400.

At the end of simulation initialization, a function of Foreign Model is called by the script. In this function, 128 signatures are generated randomly and written to the local memories of the MicroEngines with Bloom Filter Module. At the same time, these signatures are written to a log file, which can be used by the Network Traffic Simulation Plug-in.

When the simulation starts, the Network Traffic Simulation Plug-in module begins to send packets to the media bus of IXP 2400 continually. Each byte of these packets' payload is randomly generated and the length of the payload can also be changed. In order to compare the performance of packet forwarding rate, signatures are purposely inserted into the payload of packets, which should be captured by Bloom Filter. Which signature to choose is random, and the insert position in the packet is also random. By changing the length of the payload and the hit rate, the throughput in different settings is evaluated.

2) *Results*

a) *Internet Trace*

The real traces from (National Laboratory for Applied Network Research (NLANR), http://www.nlanr.net/) are used to provide realistic background traffic for simulations. 10 thousands of TCP flow headers from a 30 minutes trace are extracted and injected into the experiments. TCP flow byte streams are randomly generated and filled with padding.

128 Signatures generated randomly are randomly inserted in the payload of TCP flows. i.e., the signature may be distributed between many fragmented packets, or in an intact packet. For privacy problem, the trace data are only used in the technique issues.

b) *Throughput*

The TCP flow scanner' throughput (or Packet forwarding rate) is measured in the case that the packet is input with full rate. The throughput statistic is collected on the WorkBench 4.1. Figure 8 shows the maximum packet forwarding rate of the system with the payload size and hit ratio varying. In these experiments, the packets are simulated with length of 16, 64 and 128 Bytes, and the probability carrying the signatures is 0%, 20%, 40%, 60%, 80%, 100%.

As the system needs to scan the entire payload by shifting byte one by one, the packet forwarding rate is related to the size of the payload. When the size of payload increased, the throughput comes down slowly.

In the simulation experiment, the exact matching is also executed on ME. Once a string is matched and proved true by using exact matching, the Bloom Filter just stops and alarms without any more filter operations. Therefore, it takes less cycles when handling packets with signatures especially with long packets with high hit ratio. It means the throughput of the system will not be impaired when a worm breaks out and the network is jammed with worm's packets.

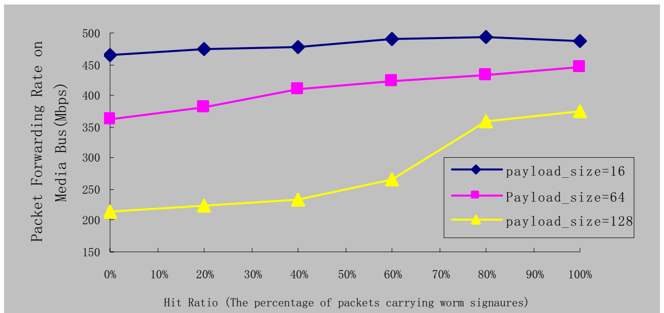


Figure 8. Packet Forwarding Rate on Media Bus with the Varied Hit Ratio

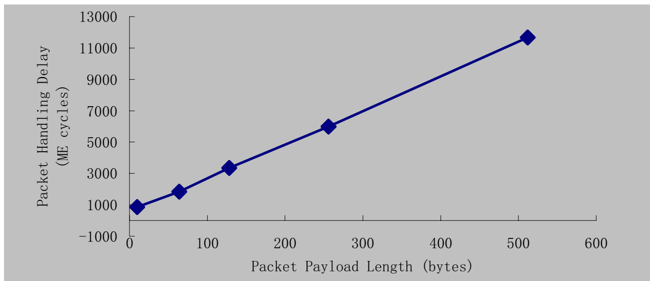


Figure 9. Relationship between packet handling delay and packet payload length.

c) *Delay*

Packet handling delay is accurately measured in the unit of cycles. The handling delay of each packet is recoded with payload varying from 64 to 1024. The packets are generated without signatures (i.e., hit ratio is 0%). It is the worse case in packet delay because the Bloom Filter must scan the whole payload. Figure 9 shows the relation between packet handling delay and packet payload length.

d) *Network Processor Internal Statistics*

The internal statistics of IXP 2400 is collected and plotted, where packet payload size is 16bytes, 64bytes and 128bytes. The statistics consists of TX ME utilization rate, Classifier ME utilization rate, Bloom Filters ME utilization rate, RX ME utilization rate, two SRAM Channel Utilization Rates, and DRAM Utilization Rate. Figures 10-13 show the internal statistics of IXP2400.

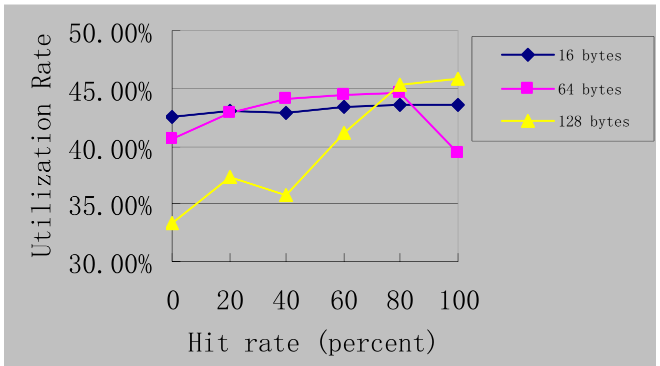


Figure 10. ME Utilization Rate (RX)

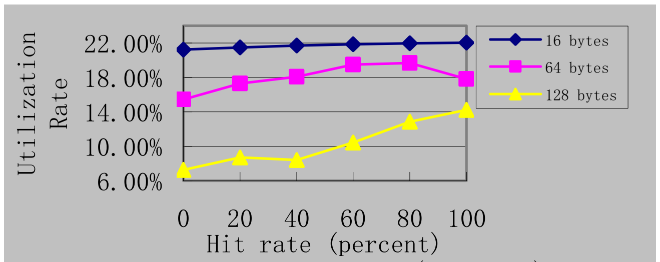


Figure 11. ME Utilization Rate (Classifier)

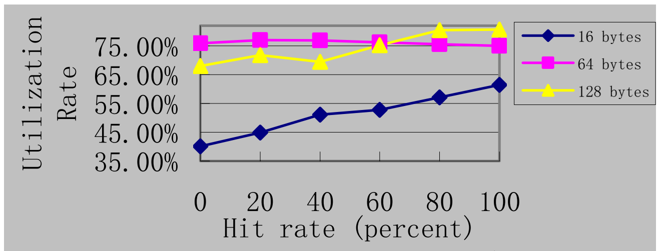


Figure 12. Average ME Utilization Rate (Bloom Filter)

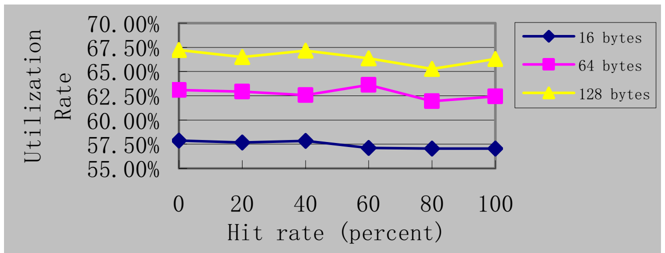


Figure 13. ME Utilization Rate (TX)

From these figures, it can be concluded that the performance bottleneck of the system is the Bloom Filters operation. The SRAM and DRAM are not overloaded because each packet only needs two read and one write operations.

B. *Real Experiments*

The real test environment is shown in Figure 14, which includes ENP-2611develope platform, Traffic Generator, several hosts and servers. ENP-2611-256 includes: IXP2400, 600Mhz, 256MB DDR SDRAM, 8MB QDR SRAM.

(1) Two independent QDR SRAM controllers, each 4MB, total 8M;

(2) 256MB DDR SDRAM;

(3) 16MB strataFlash memory to store boot code.

IXIA Traffic Generator generates TCP/IP/Ethernet packets; signatures are injected intentionally in the payload of the test packet. Another case is the signature is broken into several parts and carried by several consecutive packets in the same TCP flows. In both cases， the signatures are discovered and recorded in the Local Memory of IXP 2400.

V. Software Package

For better performance, the system modules are programmed in MicroCode (Assembly). The initialization and management procedures on XScale are written in C language. For test purpose, a traffic generator program is implemented to produce real traffic by integrating the real internet trace.

All codes have been packed into one software package, named TCPScanner 1.0, which integrates all the listed function blocks, and a signature library contains about one hundred signatures of Worms. This software package will be released as shared resource for research purpose in Network Processor field.

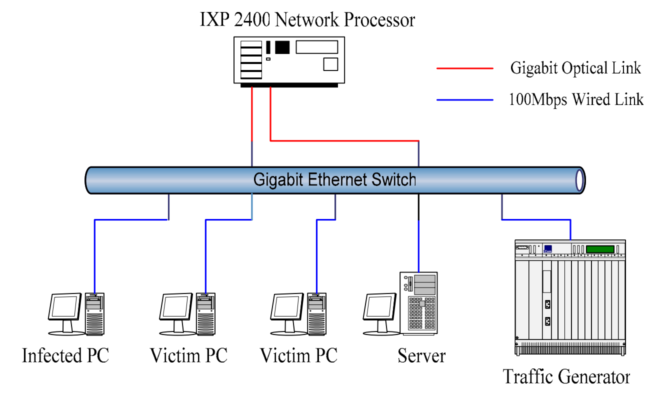


Figure 14. Gigabit Ethernet Test Environments

VI. Further Work

A. *Packet reordering*

Resequencing the out-of-order packets is very important in the system. But it has not been implemented in current versions. IXP provides the AISR for reordering the packets, but it may cost several memory spaces and handling overhead.

B. *Flow cache*

According to some statistics [26], among the large amount of active flows, few flows are large enough and arrive frequently; hence a flow state cache may be useful in accelerating lookup speed in the flow state depositary.

We will combine the ENP2611 and the main board of a PC together. In such case, the data will be offloaded to the Hard Disk directly for further inspection (scanned by other AV software, such as Symantec etc.)

C. *Polymorphic worms*

A polymorphic worm is one whose payload is transformed regularly, so no single signature identifies it. It is really a trouble for signature-based AntiWorm system to identify such worms. It is interesting to point out that [36-37] introduces signature-generated method for detecting new worm and finding the invariants of polymorphic worm.

VII. Conclusions

An AntiWorm system based on stateful TCP flow inspection scheme is implemented on Intel IXP2400 Network Processor, which can monitor about 512K flows. With the fast packet-handling and highly programmable capability provided by NPU, parallel searching engines based on Bloom Filters are implemented. The main idea is to find and locate the worm by detecting the signatures of worms in each active TCP flow byte stream. NPU-Based implementation can scan every packet with high performance and can live update the worm's signature flexibly to keep pace with fast evolution of worms.

The principles and implementation details of the system are also introduced and discussed. The performance issues about IXP Network Processors are evaluated and the statistics are collected. and an analysis is made for further optimizing the system performance. The system is also demonstrated and proved workable by using the Internet traces. An isolated test network scenario is constructed, and the real assaults of Worms are simulated by benignly executing the Worm sample codes. Software Package TCPScanner 1.0 is also given as a software release of the research.

ACKNOWLEDGMENT

We thanks for Intel Corporation for donating our Intel IXP 2400 developer platform and provides technique supports. We are grateful to Jeffery Cao in Intel China University Program for his support and also thank Brad Burres in Intel Communications Group for his insight suggestions.

REFERENCES

[1] Larry L. Peterson and Bruce S. Davie, "Computer Networks: A system Approach," second edition, Morgan Kaufmann Inc., 2000.
[2] Douglas E. Comer, "Network System Design: Using Network Processors," Pearson Education Inc., 2004.
[3] Wajdi Feghali, Brad Burres, Gilbert Wolrich, Douglas Carrigan, "Security: Adding Protection to the Network via the Network Processor," Intel Technology Journal, Volume 06, Issue 03, Published August 15, 2002.
[4] Ram Bhamidipati, Ahmad Zaidi, Siva Makineni, Kah K. Low, Robert Chen, Kin-Yip Liu, Jack Dahlgren, "Challenges and Methodologies for Implementing High-Performance Network Processors," Intel Technology Journal, Volume 06, Issue 03, Published August 15, 2002.
[5] Sridhar Lakshmanamurthy, Kin-Yip Liu, Yim Pun, Larry Huston, Uday Naik, "Network Processor Performance Analysis Methodology," Intel Technology Journal, Volume 06, Issue 03, Published August 15, 2002.
[6] Uday Naik, Alex Shoykhet, Larry Huston, Donald Hooper, Raj Yavatkar, Duke Tallam, Travis Schluessler, Prashant Chandra, Adrian Georgescu, "IXA Portability Framework: Preserving Software Investment in Network Processor Applications," Intel Technology Journal, Volume 06, Issue 03, Published August 15, 2002.
[7] Bill Carlson, Intel Internet Exchange Architecture and Applications: A Practical Guide to Intel Network Processors, Intel Press, 2003.
[8] Erik J. Johnson and Aaron R. Kunze, IXP2400/2800 Programming: The Complete MicroEngine Coding Guide, Intel Press, 2003.
[9] Peder Jungck and Simon S.Y. Shim, "Issues in High-Speed Internet Security," IEEE computer magazine, Vol. 37, No. 7, pp. 36-42, July 2004.
[10] Web Browser Security, see http://bcheck.scanit.be/bcheck/, or http://www.greymagic.com/.
[11] Darrell M. Kienzle, Matthew C. Elder, "Recent worms: a survey and trends," Proceedings of the 2003 ACM workshop on Rapid Malcode, Pages: 1–10, October 2003.
[12] David Geer, "Malicious Bots Threaten Network Security," IEEE Computer magazine, Vol.38, No.1, pp.18-20, January 2005.
[13] Signatures of worm, "Creating signatures for ClamAV," see http://www.clamav.net/doc
[14] Bill Carlson, Intel Internet Exchange Architecture and Applications: A Practical Guide to Intel Network Processors, Intel Press, 2003.
[15] Erik J. Johnson and Aaron R. Kunze, IXP2400/2800 Programming: The Complete MicroEngine Coding Guide, Intel Press, 2003.
[16] Intel Corporation, IXP2400/IXP2800 Network Processor Programmer's Reference Manuals, May 2004.
[17] Intel Corporation, IXP2400/IXP2800 Network Processor Datasheet.
[18] Intel Corporation, IXP2400/IXP2800 Network Processor Hardware Reference Manual, November 2003.
[19] Intel Corporation, IXP2400/IXP2800 Network Processor Development Tools User's Guide, July 2004.
[20] Intel Corporation, IXP2400 and IXP2800 Network Processors Packet Generator Reference Manual, July 2004.
[21] Intel Corporation, Intel Internet Exchange Architecture Software Building Blocks Developer's Manual, November 2003.
[22] B. Bloom, "Space/Time Trade-Offs in Hash Coding with Allowable Errors," Comm. ACM, vol. 13, no. 7, May 1970, pp. 422-426.
[23] Michael Mitzenmacher, "Compressed Bloom Filters," IEEE/ACM Transaction on networking, vol. 10, no. 5, october 2002.
[24] Sarang Dharmapurikar, Praveen Krishnamurthy, T.S. Sproull and J. W. Lockwood, "Deep packet inspection using parallel bloom filters," IEEE Micro, Volume 24, Issue 1, Pages:52 – 61, Jan.-Feb. 2004.
[25] D. V. Schuehler, J. Moscola and J. W. Lockwood, "Architecture for a hardware-based, TCP/IP content-processing system," IEEE Micro, Volume 24, Issue 1, Pages: 62 – 69, Jan.-Feb. 2004.
[26] Bharath Madhusudan, John W. Lockwood, A Hardware-Accelerated System for Real-Time Worm Detection, IEEE Micro, Vol. 25, No. 1, pp. 60-69, January/February 2005.
[27] J.W. Lockwood et al., "Application of Hardware Accelerated Extensible Network Nodes for Internet Worm and Virus Protection," Active Networks: IFIP TC6 5th International Workshop, IWAN 2003, Kyoto,Japan, December 10-12, 2003, Lecture Notes in Computer Science 2982, Springer-Verlag, 2003, pp. 44-57.
[28] J.W. Lockwood, "An Open Platform for Development of Network Processing Modules in Reprogrammable Hardware," Proc. IEC DesignCon 01, Int'l Eng. Consortium, 2001, pp. WB-19.
[29] F. Braun, J. Lockwood, and M. Waldvogel, "Protocol Wrappers for Layered Network Packet Processing in Reconfigurable Hardware," IEEE Micro, vol. 22, no. 1, Jan. 2002, pp. 66-74.
[30] Cristian Estan, George Varghese, New Directions in Traffic Measurement and Accounting, ACM SIGCOMM 2002.
[31] C. Estan and G. Varghese, "New Directions in Traffic Measurement and Accounting," Proc. 2002 Conf. Applications, Technologies, Architectures, and Protocols for Computer Comm., ACM Press, 2002, pp. 323-336.
[32] C. Estan, G. Varghese, and M. Fisk, "Bitmap Algorithms for Counting Active Flows on High-Speed Links," Proc. 3rd ACM SIGCOMM Conf. Internet Measurement, ACM Press, 2003, pp. 153-166.
[33] A. Kumar et al., "Data Streaming Algorithms for Efficient and Accurate Estimation of Flow Size Distribution," Proc. Joint Int'l Conf. Measurement and Modeling of Computer Systems, ACM Press, 2004, pp. 177-188.
[34] N. G. Duffield, C. Lund, M. Thorup, Estimating flow distributions from sampled flow statistics, ACM SIGCOMM 2003.
[35] N. Brownlee, C. Mills and G. Ruth, Traffic flow measurement: Architecture, RFC 2722.
[36] James Newsome, Brad Karp, Dawn Song, "Polygraph: Automatic Signature Generation for Polymorphic Worms," IEEE Symposium on Security and Privacy, May 2005.
[37] James Newsome and Dawn Song, "Dynamic Taint Analysis: Automatic Detection, Analysis, and Signature Generation of Exploit Attacks on Commodity Software," Network and Distributed Systems Security Symposium, Feb 2005.
[38] D. Moore et al., "Internet Quarantine: Requirements for Containing Self-Propagating Code," Proc. IEEE Infocom2003, IEEE Press, 2003, pp. 1901-1910.
[39] S. Singh et al., The EarlyBird System for the Real-Time Detection of Unknown Worms, tech. report CS2003-0761, Dept. of Computer Science, Univ. of Calif., San Diego, Aug. 2003.
[40] J. Moscola et al., "Implementation of a Streaming Content Search-and-Replace Module for an Internet Firewall," Proc. 11th Symposium on High-Performance Interconnects, IEEE CS Press, 2003, pp. 122-129.
[41] George Bakos and Drs. Vincent H. Berk, Early Detection of Internet Worm Activity Using ICMP Destination Unreachable Messages, In Proceedings of the SPIE Aerosense, 2002.
[42] G. Gonnet and E. Brewer, Handbook of Algorithms and Data Structures, Addison- Wesley, 1991.