TCP Reassembly For Signature-based Network Intrusion Detection Systems

Tran Ngoc Thinh
Dept. of Computer Engineering
Faculty of Computer Science and Engineering, HCMUT
Ho Chi Minh city, Vietnam
tnthinh@cse.hcmut.edu.vn

Surin Kittitornkun
Faculty of Engineering
King Mongkut's Institute of Technology
Lardkrabang, Thailand
kksurin@kmitl.ac.th

Shigenori Tomiyama
School of Information Telecommunication Engineering,
Tokai University
Kanagawa-ken, Japan
tomiyama@keyaki.cc.u-tokai.ac.jp

Tran Huy Vu
Dept. of Computer Engineering
Faculty of Computer Science and Engineering, HCMUT
Ho Chi Minh city, Vietnam
vutran@cse.hcmut.edu.vn

***Abstract*—Rapid development of network makes it a very important and vulnerable part of every field of life. Many intrusion detection systems are developed to protect the network using signature-based matching technique. For connection oriented protocols, such as Transmission Control Protocol, the data should be reassembled before being scanned by the matching engine. Several techniques are introduced to reassemble TCP packets on FPGA. However, they have some disadvantages such as inefficient memory, unscalable system, and unsupported complex TCP connections. In this paper, we propose a multi-linked-list approach and a combination of edge buffering scheme for TCP reassembly, which helps detecting cross packets intrusion signatures. Our architecture not only supports TCP connections with up to 4 concurrent holes, but also uses memory more efficiently than others. The experimental results show that our system can hold about 256K connections simultaneously and support up to 46K out-of-sequence connections with only 64MB DRAM.**

Keywords- TCP reassembly, Segment array, Linked list, Edge, FPGA

I. INTRODUCTION

Nowadays network is vital to almost every organization. That is why many information crimes are network intrusions. These intrusions are based on many types of protocols. However, the Transmission Control Protocol (TCP) is the most popular, the authors in [12] showed that more than 90% of network traffic is TCP. In TCP protocol, the data is split into packets, and these packets are transmitted consequently. However, they can arrive at the destination in the wrong order due to transmission error or the network routing mechanism. There exists some Network Intrusion Detection / Prevention Systems (NIDS/NIPS) to prevent attacks on an organization. If the intrusion patterns are included in a single packet, they can be detected by traditional NIDS/NIPS; but if the intrusion patterns expand over packets, and these packets do not arrive in the original order (out-of-sequence), they cannot be detected as described in Figure 1. Moreover the network speed can reach 1Gbps or more, these NIDS systems can drop the packets. Studies [4] show that full TCP reassembly requires large amount of memory, up to 2GB for each connection. To overcome these problems, several researches have been proceeded to reassemble out-of-sequence TCP packets [1, 3, 4, 5]. In these researches, FPGA is usually used as the development platform because it supports high speed processing and easy to update. The TCP Processor in [4] uses the retransmission mechanism to reorder the out-of-sequence packets. It drops all out-of-sequence packets so that the source terminal will retransmit all packets that have not come in the right order. The advantages of this approach are simplicity, memory saving, but it causes the network traffic to be heavier, and may prevent the destination terminal from efficient acknowledgement. Moreover, though the percentage of out-of-sequence connection is little, these connections are usually long connections; the number of retransmitted packets can be very large. The authors chose this approach because a statistical result in [7] shows that only about 5% of TCP packets are out-of-sequence. Another simple technique is introduced in [3], which do not need to buffer a whole packet. It has to buffer only data at 2 ends of a packet, the length of buffered data equals the maximum length of rules. This technique does not require to reorder packets, because packets are processed overlapping together. However, this technique can only be applied to static scanning engine because the maximum length of a pattern is priory known. In practice, many applications use both static pattern and regular expression (RE). The length of a string, which matches an RE, cannot be priory known. In another approach [1], the authors use a buffer for each out-of-sequence connection. The size of the buffer is fixed and every out-of-sequence connection has only one buffer. If an out-of-sequence TCP packet comes, its

978-1-4673-2025-2/12/$31.00 ©2012 IEEE

sequence number is used to compute the offset from the start of the buffer to store the packet. This method is not efficient because a large packet cannot be contained in a buffer, but a tiny packet can waste a lot of memory in the buffer. This method may require a lot of memory, and does not support large number of concurrent out-of-sequence connections. Using the similar approach of buffer, the TCP reassembly in [5] uses a linked list to store out-of-sequence packets, but the control information of a packet is store in SRAM. Moreover, the data structure of reassembly memory is a linked list of separate packets; this structure is not memory-efficient. This system has to reserve maximum DRAM space, for example 1500 bytes, for each out-of-sequence packet, so it can waste a lot of DRAM memory. This system supports quite few connections simultaneously. Also use the linked-list approach, Sarang Dharmapurikar and Vern Paxson in [2] limited the number of holes in a connection to only one hole. They use linked list to store out-of-sequence packets, which are all stored in DRAM. In this system, the memory is divided into blocks; each packet can be stored in more than 2 blocks, 1 block can contain more than 2 packets, so the memory utilization is more efficient. The reason that they decide to supports only one hole is based on their studies that more than 95% of out-of-sequence connections contain only one hole. However, newer studies in [1] show that number of 2-hole connections are 7.3% of out-of-sequence connections in CAIDA_10G, and even 17.8% in WA_1G. The percentage of multi-holes connections is considerable, it is necessary to covers these connections. In the next section, we propose another method that uses the memory efficiently, support multi-hole connections, and operates in Giga-bit network.

II. MULTI-LINKED-LIST APPROACH

In the above approaches, the design in [2] shows its efficient memory utilization, but it has the disadvantages as stated above. In our design, we also use one linked list to store the payload of packets in the same segment. Because the data in the same segment are consecutive, they are suitable to be stored in a linked-list of memory blocks. A modification of edge buffering scheme [3] is also applied to this system. To support connections with more than 1 hole, each out-of-sequence connection uses an array to manage several holes. Each array element has a pointer to refer to a segment, so the array size is the maximum concurrent holes in a connection. Studies [1] show that about 99% out-of-sequence connections have less than 4 concurrent holes. Moreover, if a connection has too many holes, buffering all out-of-sequence packets is not as efficient as dropping the packets so that the source machine will retransmit packets to fill some holes. Therefore, in our design, we limit the number of concurrent holes in a connection to 4, and thus the array size is 4. When an out-of-sequence packet arrives at the system, there are four cases to manipulate the segment array: allocating a new linked list, inserting packet payload to an existing linked list, merging two existing linked lists, and releasing an existing linked list. However, if an out-of-sequence packet makes the number of concurrent holes in the corresponding connection exceed the size of the segment array, the packet is dropped.

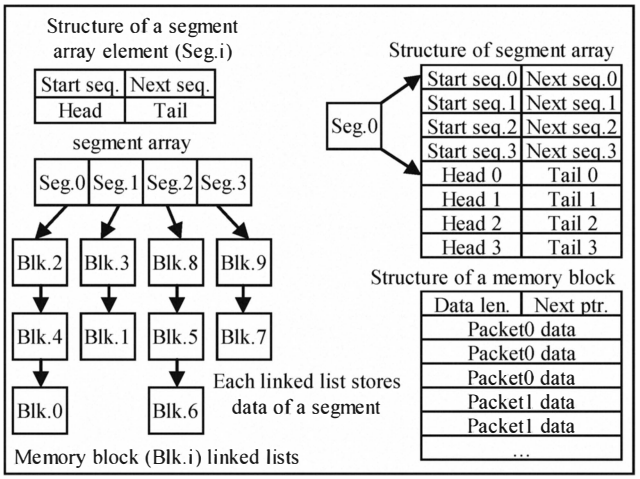

Figure 1 Data structure of reassembly memory

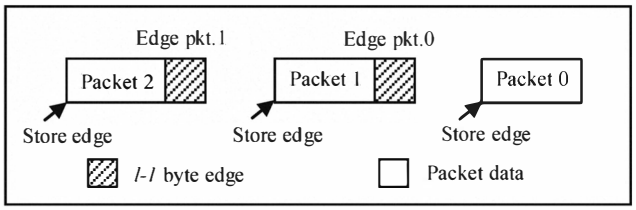

Figure 2 Edge buffering scheme applying to ordered TCP packets

Source IP		Dest. IP
Source Port	Dest. Port	Sequence
Flags	Buffer address	App. FSM

Figure 3 A Connection record (CR)

Figure 1 describes the data structure of the out-of-sequence buffer. Each segment array element contains following information. *Start seq.* is the sequence number of the first byte of the segment. *Next seq.* is the next expected sequence number of the segment. *Head* is address of the fist byte of the segment in DRAM. *Tail* is address of the last byte of the segment in DRAM. For fast retrieving of each segment, we construct the segment array in a special way. All sequence fields are consecutive, all address fields are consecutive, and so the sequence fields and the address fields of a segment are not consecutive. We use this data structure because we want to arrange the sequence fields so that many sequence fields can be read in one DRAM access. Moreover, the system can operate on the sequence fields and the address fields separately. When a packet comes, the first stage of the system accesses sequence fields quickly, determines the action to be issued, and requests the second stage to proceed the action. This operation can be done in one DRAM access. The real operation on the packet payload, which can require much time, is carried out by the second stage independently. Because of this independence, the reassembly operation is pipelined, and does not impact on the throughput of the system. Another issue is to manage the data in a segment efficiently. We divide memory space into blocks; the structure of a block is simple as in Figure 1. *Data len.* is the number of valid data bytes stored in the block. *Next ptr.* is the address of next block in the same linked list. The Data of each

segment is stored in a linked list of blocks. The address of the linked list and the *Next ptr* are not necessary aligned with the block address; this situation occurs when the payload of a packet is pre-pended to an existing linked list. If the payload of a packet is larger than the block size, it is stored in 2 or more blocks. If the payload of a packet is smaller than the block size, the next packet in the same segment can fill in that block.

In pattern matching systems, there are matching engines which do not deploy an explicit Finite State Machine (FSM). For example, some NIDSes use hashing methods, such as Bloom filter [10] or Cuckoo hashing [9], to match static patterns. In such situations, the system cannot reassemble packets by loading and storing FSM of the matching engine. In this paper, we apply the idea of One-edge and Two-edge buffering scheme [3] to store the overlapped data between packets. However, TCP packets are ordered by using multi-linked-list method, the edge buffering scheme can be simplified as in Figure 4. When the packets are ordered, only the ending edge of each packet is needed to be stored and re-scanned. Though the system can store a long ending edge to cover all patterns, the system has to re-scan a lot of data, and thus the throughput of the system is decreased. Applying both multi-linked-list method and edge buffering scheme, we implement a TCP reassembly system for our NIDS on FPGA as below.

To manage the status of each connection, a Conection Record (CR) is used; the detail of a CR is described in Figure 3. Each TCP connection is identified by 4 fields: *Source IP*, *Dest. IP*, *Source Port*, *Dest. Port*. All packets have the same value of these fields belong to the same connection and the *Sequence* number indicates the order of the packet in the connection. The *Flags* include SYN and ACK flags to manage 3-way hand shaking scheme in TCP connection establishment; another flags is EST indicates the record is occupied. *Buffer address* is a pointer to refer to a segment array of the connection. For matching engines using an algorithm with an explicit FSM, such as Aho-Corasick [8], the FSM is stored to/ loaded from the *App. FSM.*

III. IMPLEMENTATION

The Figure 3 is our implementation of the proposed method. When a packet arrives, it is buffered in a FIFO to wait for the checking procedure. The system calculates the hash value of 4 fields *Source IP*, *Dest. IP*, *Source Port*, *Dest. Port*, and uses this value as the address to access the CR. These 4 fields are compared to the 4 fields of the CR to determine whether the packet belong to the current CR. The sequence number is then used to determine the order of the packet.

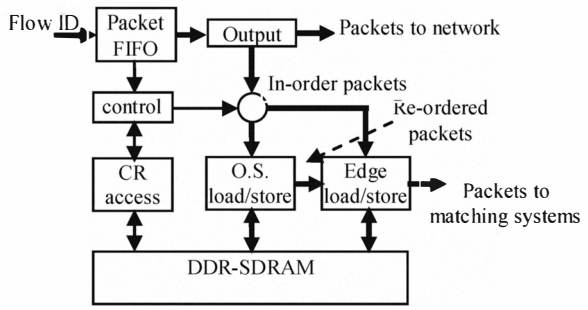

Figure 4 The TCP reassembly using the proposed method

When the first SYN packet arrives, if the corresponding CR is available (EST flag is set to 0), the CR is updated with the new connection. If the system receives a FIN packet, the EST flag of the corresponding CR is cleared. If the packet is in-sequence, its sequence number is updated to the CR, and then the packet is sent both to the network and the matching engine. However, it is prefixed with the ending edge of the last packet before entering the matching engine. The system also checks the out-of-sequence buffer if there is any data that is consecutive with the packet. If there is any such data, it is read out and sent to the matching engine as a suffix of the packet. The system then makes an update to the segment array because of the removal of a segment. A retransmitted packet is already processed by the system, and it may be dropped on the link between the system and the destination. In this case, the packet is sent to the network only, and there is no update on the CR If the packet is an out-of-sequence packet, it is stored to the out-of-sequence buffer using multi-linked-list method as in previous section. The system also updates the corresponding segment array to reflex the new size of a segment. Because the reassembly memory needs a large amount of memory, it is resided in DRAM. The segment arrays and the memory blocks are dynamically allocated and released. However, the edge data is not dynamically allocated. Because every connection needs an edge buffer, it occupies 32 bytes next to the CR.

IV. EVALUATION

To evaluate our approach, we synthesis our design as described in section 4 on Virtex2-Pro FPGA chip. We use Xilinx ISE 10.1i for synthesis and timing simulation. The synthesis results show that our design can operate at clock rate of 157MHz. The number of occupied slices is about 18% of total slices of Virtex2-Pro. So our TCP reassembly system can be fully integrated into Virtex2-Pro along with an application circuit such as NIDS. Our system uses 16MB DRAM to store 256K connection records. It uses 46MB DRAM to store out-of-sequence packets with the block size is 1KB, so it can hold maximum 46K out-of-sequence connections. It also uses 2MB DRAM to store up to 64K segment arrays. The total used memory is 64MB.

We also compare our system to other systems. We do not compare our system with the system in [3] because that system use the method of out-of-order matching, and thus do not need to reorder out-of-sequence packets. We assume the network traffic conforms to the CAIDA_10G in [1]. These statistics are recorded by the Cooperative Association for Internet Data Analysis in 2009.

Table I Number of holes supported by our system and other systems

Number of holes	Percentage	Fixed buffer [1]	1-hole linked list [2]	Simple linked-list [5]	Our system
1	89.6%	support	support	support	support
2	7.3%	support		support	support
3	1.9%	support		support	support
≥4	1.2%	support		support	support
Total	100%	100%	89.6%	100%	>98.8%

Table II Memory utilization of our system and other systems for buffering 64k single-hole connections

Number of packets	Fixed buffer [1]	1-hole linked list [2]	Simple linked-list [5]	Our system
1	1024MB	128MB	93.75MB	68MB
2	1024MB	128MB	187.5MB	68MB
3	1024MB	128MB	281.25MB	132MB
4	1024MB	128MB	375MB	132MB

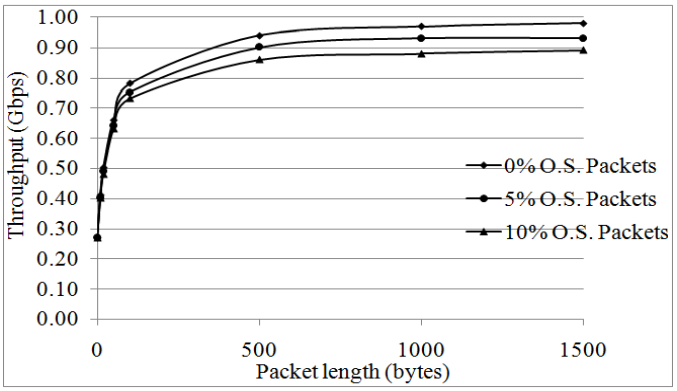

Figure 5 Throughputs of the system

Table I shows that our system can support 96.9% of out-of-sequence connections compared to 89.6% in system in [2]. On the other hand, our system can easily scale up to support 4-hole connections; in that case the ratio of supported out-of-sequence connections can exceed 98.8%. Theoretically, the fixed length buffer method [1] and simple linked list method [5] can support more than 4 concurrent holes in a single connection. However, the number of connections with more than 4 holes is very small, so dropping packets which create more than 4 holes in a connection is more practical.

In Table II, we compare the memory utilization of our system with other systems. Because the design in [2] supports only single-hole connections, we compare the memory utilization for single-hole connections only. The first column is the number of out-of-sequence packets in a single-hole connection. Based on the statistical data in [1], the mean packet size is 441 bytes. In [1], each out-of-sequence connection has a fixed length buffer. Based on the experimental result of the authors, the minimum size of buffer is 16KB, so buffering 64K connections requires 1024MB of memory. In [2], the page size is 2KB, so the memory requirement is calculated as in Table II. If this system uses the page size of 1KB, the memory requirements is a little smaller than the memory requirement of our system, about 4MB. However, this system cannot handle connections with more than 1 hole. In [5], the system uses linked list of packets, so the system has to reserve a large space enough to store a biggest packet. The results show that our system uses memory more efficiently than the others.

Figure 5 shows the throughput of our system. The throughputs depend on the packet lengths, percentage of out-of-sequence packets, as well as the edge length to be stored. These numbers are measured with the edge length of 32 bytes. The packet lengths are chosen from 100 to 1500 (maximum Ethernet packet length). The typical percentages of out-of-sequence packets are from 0% to 10%. The experiments are made on NetFPGA 1G board [11] with 1Gbps Ethernet interface. The results show that our system can operates on Gigabit network.

V. CONCLUSION

In this paper, we present a technique of TCP reassembly. We focus on multi-linked list method to manage the memory of out-of-sequence packets, which is efficient and easy to scale up in the future. Besides, the edge buffering scheme is also deployed to detect cross-packet intrusion patterns in signature scanning engine. Experimental results show that our system can hold about 256K concurrent connections and 46K out-of-sequence connections with only 64MB DRAM. Our architecture supports connections with multiple concurrent holes, and it can support up to 99% of out-of-sequence connections. Moreover, if we need to change the system to support more concurrent holes in a connection, we just need to change the size of the segment array.

REFERENCES

[1] Ruan Yuan, Yang Weibing, Chen Mingyu, Zhao Xiaofang, Fan Jianping – Robust TCP Reassembly with a Hardware-based Solution for Backbone Traffic, the Fifth IEEE International Conference on Networking, Architecture and Storage, 2010, pp. 439-447

[2] Dharmapurikar, S., & Paxson – Robust TCP Reassembly in the Presence of Adversaries, Proceedings of the 14th conference on USENIX Security Symposium Volume 14, 2005, pp. 65–80.

[3] Sugawara Y., Inaba M., Hiraki, K. – High-speed and Memory Efficient TCP Stream Scanning Using FPGA, Field Programmable Logic and Applications International Conference, 2005, pp 45–50.

[4] David V. Schuehler - Techniques for Processing TCP/IP Flow Content in Network Switches at Gigabit Line Rates, Doctoral Dissertation, December 2004.

[5] Palak Agarwal – Tcp Stream reassembly and web base gui for Sachet IDS, Master Thesis, Department of Computer Science and Engineering, Indian Institute of Technology, Kanpur, India, 2007.

[6] Hao Chen, Yu Chen, Douglas H.Summerville – A survey on the Application of FPGAs for Network Infrastructure Security, the IEEE Communications Surveys and Tutorials, 2010, pp. 1-21.

[7] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley. Measurement and classification of out-of-sequence packets in a tier-1 IP backbone. Technical Report CS Dept. Tech. Report 02-17, UMass, May 2002, pp. 54-66.

[8] Derek Pao, Wei Lin, Bin Liu, A memory-efficient pipelined implementation of the Aho-Corasick string matching algorithm, TACO 2010.

[9] Tran Ngoc Thinh, Kittitornkun S., Tomiyama S. – Applying Cuckoo Hashing for FPGA-based Pattern Matching in NIDS/NIPS, International Conference on Field Programmable Technology, 2007, pp. 121-128.

[10] Bui T. Hieu, Nguyen D. A. Tuan, and Tran N. Thinh, BBFex: An Efficient FPGA-based Design for Long Patterns in Pattern Matching System. *2010 International Conference on Intelligent Network and Computing (ICINC 2010)*, Nov 26-28, 2010, Kuala Lumpur,

[11] http://www.netfpga.org/php/specs.php.

[12] Basem Shihada and Pin-Han Ho, Transport Control Protocol in Optical Burst Switched Networks: Issues, Solutions and Challenges. The IEEE Communications Surveys and Tutorials, 2nd Quarter 2008, Volume 10, No.2