Session Management Architecture for Implementing an FPGA-based Stateful Intrusion Detection System

Seungyong Yoon, Byoungkoo Kim, Jintae Oh, and Jongsoo Jang
Security Gateway System Team, Electronics and Telecommunications Research Institute
161 Gajeong-dong, Yuseong-gu, Daejeon, 305-700
Republic of KOREA

Abstract: - This paper relates to session management architecture and mechanism for implementing an FPGA-based stateful intrusion detection system. Our proposed architecture can help to perform Stateful Packet Inspection(SPI) in real time using a new session table management scheme that allows more efficient generation of session state information in intrusion detection system. SPI is an important technique to reduce false positive alerts in network intrusion detection system(NIDS). As the number of session increases, this technique requires a higher processing speed, thereby causing performance problems. However, existing software-based solutions cannot perform real-time packet inspection ensuring the wire speed. To guarantee both performance and functionality with respect to statefulness, we designed and implemented SPI-based intrusion detection module in a FPGA to help alleviating a bottleneck in network intrusion detection systems in this paper.

Key-Words: - Intrusion Detection, Stateful Packet Inspection

1 Introduction

One of the major problems and limiting factors with Network Intrusion Detection System(NIDS) is the high false positive alert rate. In order to reduce these false positive alerts, a lot of methods and techniques are proposed. Stateful Packet Inspection(SPI) is one of these solutions. SPI was originally developed for Firewall[1][2], but it became a very important factor in NIDS. Stateless NIDSs generate tremendous false positive alerts while stick or snot attempts to attack[3][4]. Most existing NIDSs have SPI module which is supported statefulness but they don’t satisfy high-performance in gigabit internet environment. It is so difficult that we manage a lot of session state information with limited hardware resource and satisfy performance of high-speed internet. In other words, the rapid evolution of recent network technologies to gigabit network environments require existing SPI module to have more improved functions and performance. SPI basically requires a session table which stores source and destination IP addresses and port numbers. It is necessary to perform real-time packet inspection by checking, for each input packet, whether or not a corresponding entry is present in the session table. Real-time packet processing at wire speed should not cause any packet delay or loss even when the number of managed sessions is increased to more than one million.

Previously developed software-based solutions cannot meet these requirements. One software-based technique has attempted to use a distributed system. However, as the number of session increases, this technique requires a higher processing speed, thereby causing performance problems. Thus, software-based solutions cannot perform real-time packet inspection ensuring the wire speed.

To guarantee both performance and functionality with respect to statefulness, we designed and implemented SPI-based intrusion detection module in a FPGA to help alleviating a bottleneck in network intrusion detection systems in this paper. The performance of SPI-based intrusion detection system mainly depends on the performance of processing session table and pattern matching[5][6]. In this paper, we focused on session state management scheme and omitted pattern matching method. Our work related to pattern matching method is described by Byoungkoo Kim at al. in detail[7].

2 SPI-based Intrusion Detection Module

Our SPI-based intrusion detection module was implemented on Security Gateway System(SGS). SGS is a security node system which is located at ingress point in protected network. Strictly speaking, SGS is network intrusion prevention system running in inline mode. Fig.1 shows the SPI-based intrusion detection module of our SGS. Legitimate TCP sessions are established through 3-way handshake and terminated through 4-way handshake. State manager has session table and tracks these session state. If input packet doesn’t exist in session entries, this packet will drop or forward to Intrusion
Detection Engine (IDE) with additional state information according to security policies.

At first, if the packet inputted from IP De-fragmentation sub-module, necessary information fields are extracted through packet parsing sub-module. Packet filter transfer to state manager only packet that passed by security filtering policies.

3 Session Management Architecture

3.1 Basic Architecture

The terms “session”, “connection” and “flow” are used interchangeably in this paper. Fig. 2 is a basic architecture of session state manager for stateful packet inspection. As shown in Fig. 2, session state manager includes a hash key generator, a session table, a session detection module, a session management module, and a state info generation module.

The session table stores session entries that are indexed and managed by the hash key generator. The 4-tuple information including a source IP address, a destination IP address, a source port, and a destination port is input, as information used to hash a newly received packet, to the hash key generator. Once the packet is inputted, a packet parser extracts this information from the packet. The hash key generator indexes and manages a session entry corresponding to the received packet based on the input 4-tuple information. Hash key generator has a dual hash structure with two different hash functions Hash1(x) and Hash2(x). The hash functions Hash1(x) and Hash2(x) are well-known functions that are used to hash packets. For example, XOR or CRC functions can be used as these hash functions. One hash function “Hash1(x)” is used to generate indices that point to hash sets permitting hash collisions in order to achieve faster session table search. The other hash function “Hash2(x)” is used to generate hash addresses that are used to identify session entries in a hash set pointed by the hash function “Hash1(x)”. Session table may be designed and implemented using two or more SRAM devices, if necessary. In this paper, the session table is constructed using two SRAMs (SRAM#1 and SRAM#2), which can be accessed simultaneously or in parallel using a hash set index that is generated by the Hash1(x) to achieve faster session table search. The session table stores session data of packets inputted from an external network. For efficient session table management, the session table has an N-way set associative session table structure in which each hash set in the session table can include N session entries. The session table shown in Fig. 2 is a 32-way set associative session table that is constructed using two 72-Megabit SRAMs with each session entry having a length of 36 bits.

Each session entry stored in the session table includes current state, time stamp, and hash address parts. The current state part includes current connection state information of a corresponding session, the time stamp part is used to determine which session entry is to be deleted when the session table is full, and the hash address part is used to identify each session entry in the same hash set. The time stamp is updated by an internal timer each time a corresponding session is accessed. If any hash sets of the session table are full so that new session cannot be allocated to the hash sets, current time of internal timer is compared with the time stamp of each session entry to replace the oldest session with a new session. For example, Least Recently Used (LRU) algorithm is applied to this process.

Session state information is separately managed in embryonic state and established state for timeout mechanism. Embryonic state includes sessions that TCP 3-way handshaking does not finish, on the other hand established state includes completed sessions. It is necessary to manage separately states, because embryonic session needs to have shorter timeout value than that of established session. The SPI devices and computers have vulnerabilities to SYN flooding attack in nature[8][9]. This mechanism helps to prevent against denial-of-service attack such as SYN flooding.

Although a Transmission Control Protocol (TCP) session is terminated without sending an RST or FIN
packet, a corresponding session entry is immediately removed if a time stamp in the session entry exceeds a timeout threshold predetermined by the administrator. Accordingly, a session which has been terminated without sending an RST or FIN packet is positively removed from the session table.

The session detection module searches the session table according to the received packet. Specifically, the session detection module obtains a hash set pointer from Hash1(x) calculated by the hash key generator and then searches the session table for a session entry corresponding to the hash value from Hash2(x). The session management module performs a process for adding, deleting, and changing sessions of the session table in order to maintain the session table. The state info generation module generates state information regarding the direction of the packet and session establishment information and then transmits this information to intrusion detection engine with packet data.

Fig. 3 schematically describes a method for processing direction information included in each packet, which indicates the direction of the packet in a corresponding session. Each packet transmitted over the network includes information regarding the direction of the packet in a corresponding session, which indicates whether the direction of the packet is from the client to the server or from the server to the client. This information is very useful in a network intrusion detection or prevention system. However, the direction information may cause a significant confusion in searching for a corresponding session in the session table since the hash address of each packet belonging to the same session may vary depending on the direction. To prevent the hash address from varying depending on the direction, the hash key generator compares the value of a source IP address with the value of a destination IP address and modifies a corresponding 4-tuple value so that one of the source and destination addresses, which has the lower value, always precedes the other with the higher value. A specific flag is defined to indicate whether or not such a position change has been made. For example, a flag “Position_change_flag” is defined, which is assigned “1” when the position change has been made and “0” when no position change has been made. The “Position_change_flag” information is very efficiently used in generating state information together with current state information.

### 3.2 Session State Information

Session state information is stored in a current state part in each session entry. In consideration of hardware resource, we designed current state part to have 3 bits in a 36-bit session entry. The first bit of the current state part contains session establishment information. For example, when a session has been established between the client and the server, the first bit is set to “1” and, when no session has been established between the client and the server, the first bit is set to “0”. The second bit of the current state part contains information indicating whether or not the source and the destination were reversed when the session was registered in the session table. This information is different from the information contained in the flag “Position_change_flag” shown in Fig. 3. The difference between the information contained in the second bit and the information contained in the flag “Position_change_flag” is described below in detail. The third bit of the current state part contains information indicating whether or not the connection is in a half-closed state. Each session is terminated only when the second FIN packet is received when the connection of the session is in a half-closed state. Namely, when the connection is in a half-closed state, the third bit is set to “1” and, when the connection is not in a half-closed state, the third bit is set to “0”. The TCP connection establishment process is performed through 3-way handshake. When the client sends a SYN packet to the server to request it to establish a new connection, the server responds with a SYN/ACK packet and then the client sends an ACK packet to the server in response to the SYN/ACK packet, thereby completing the establishment of the connection. The TCP connection termination process is normally performed through an RST packet or an FIN packet. The FIN packet is transmitted through 4-way handshake. If one of the client and the server sends an FIN packet, then the other sends an ACK packet in response to the FIN packet. This state in which the first packet has been received is referred to as a “half-closed state”. If the client transmits a second FIN packet in the half-closed state, then the server transmits a second ACK packet in response to the second FIN packet, thereby terminating the TCP session.

Fig. 4 is a state transition diagram showing the relationship between the 3-bit values stored in the current state part of session entry and input packet. The current state value is “000” in an initial state where no session has been established between the client and the server. If the client transmits a SYN packet to the server to establish a TCP session, the current state value transits to “001”. Thereafter, if a SYN/ACK packet is transmitted, the
current state value transits to “010”. If the last ACK packet is transmitted in the state of “010” in the 3-way TCP handshake process for establishing a TCP connection, the value of the source is compared with the value of the destination. The current state value transits to “110” if the position change has been made. However, the current state value transits to “100” if no position change has been made. If the first FIN packet for terminating the TCP connection is transmitted in the “110” state, the current state value transits to “111”. Thereafter, if the second FIN packet is transmitted in the “111” state, the current state value transits to the initial state value “000”. If the first FIN packet is transmitted in the “100” state, the current state value transits to “101”. Thereafter, if the second FIN packet is transmitted in the “101” state, the current state value transits to the initial state value “000”. If an RST packet for terminating the TCP connection is transmitted in any one of the “110”, “100”, “101”, and “111” states, the current state value transits to the initial state value “000”.

![Session State Transition Diagram](image)

**Table 1. State Information Generation**

<table>
<thead>
<tr>
<th>Position_change_flag</th>
<th>Current State</th>
<th>State Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any Value</td>
<td>000</td>
<td>Not Established</td>
</tr>
<tr>
<td>Any Value</td>
<td>001</td>
<td>SYN_RCVD(3-way Handshaking)</td>
</tr>
<tr>
<td>Any Value</td>
<td>010</td>
<td>SYNACK_RCVD(3-way Handshaking)</td>
</tr>
<tr>
<td>Any Value</td>
<td>011</td>
<td>Reserved(unused)</td>
</tr>
<tr>
<td>0</td>
<td>100,101</td>
<td>Established, Direction: Client → Server</td>
</tr>
<tr>
<td>1</td>
<td>100,101</td>
<td>Established, Direction: Server → Client</td>
</tr>
<tr>
<td>1</td>
<td>110,111</td>
<td>Established, Direction: Client → Server</td>
</tr>
<tr>
<td>0</td>
<td>110,111</td>
<td>Established, Direction: Server → Client</td>
</tr>
</tbody>
</table>

Fig. 4. Session State Transition Diagram

Table 1 shows state information generated from a “Position_change_flag” value and a current state value stored in the current state part. The state information is generated basically using the current state value and the direction of each packet is determined from a combination of the current state value and the “Position_change_flag” value. For example, if the current state value is “100” or “101” while the Position_change_flag value is “0”, the direction of the current packet is from the client to the server since the source and destination of the current packet have not been reversed and the source and destination had not been reversed (i.e., the direction was from the client to the server) when the corresponding session was registered. On the other hand, if the current state value is “100” or “101” while the Position_change_flag value is “1”, the direction of the current packet is from the server to the client since the source and destination of the current packet have been reversed and the source and destination had not been reversed (i.e., the direction was from the client to the server) when the corresponding session was registered.

SPI module has a following procedure for processing a packet according to our proposed scheme. First, when a packet is inputted, a hash key value is generated using 4-tuple information extracted from the packet and a session table is searched for a corresponding session using the generated hash key value. If the corresponding session is found in the session table, its session entry information is updated. If the corresponding session is not found in the session table, a new session is generated only when the current packet is a SYN packet. If the session table is full, the oldest session entry is selected using the LRU algorithm and then replaced with the new session. Once the session table for the received packet is constructed as described above, state information of the packet is generated. It is preferable that the method described in Table 1 be used to generate the state information of the packet. Then, inspection of the packet is performed based on the generated state information.

### 3.3 Performance Simulation

Our session state management scheme in SPI-based intrusion detection system is affected by two major factors, hash collision rate and miss rate. There is every probability of hash collision occurrence because hash function for faster session table search is used. The wrong state information is generated if the hash collision is occurred. Therefore, the SPI-based intrusion detection module generates the false positive alert. The hash collision rate is determined by the Hash 1(x) and Hash 2 (x). Theoretically, the probability of hash collision is \(1/2^{43}(\text{Hash 1(x)}: 18 \text{ Bits} + \text{Hash 2(x)}: 25 \text{ Bits} = 43 \text{ Bits})\).

As the number of session entries increase gradually, the session table is filled with new session. Also, there is every probability of miss occurrence because the size of hash set has limitation(32-way set). When the session table is full, the probability that each session is missed is very important in a session table management scheme.
because wrong session state information is generated if any existing session, which has not yet been terminated, is replaced with a new session. In this case, since the SPI-based intrusion detection module generates the false negative alert, the miss rate can be said to be the factor which is important than the hash collision rate. In order to ensure that miss rate is reasonable in our design, we made a simulation for distribution of the number of sessions allocated to each hash set in the session table when one million sessions are established. We used a separate set of traffic data collected from various network environments for this simulation. Fig. 5 shows the result of this simulation.

Distribution of the number of sessions allocated to each hash set in the session table follows a normal distribution as expressed by Probability density function (Equation (1)). This is standardized using Equations (2) and (3) and then the miss probability of each session in the 32-way set associative session table is calculated to obtain \( P\{X>32\} = P\{Z>8.3\} \). This indicates Z-score of 8.3 which is nearly 0%. (Z-score of 6 corresponding probability is 0.0003%).

\[
(1) \quad f(x) = \frac{1}{\sqrt{2\pi}} \exp\left\{ -\frac{1}{2\sigma^2}(x-\mu)^2 \right\}
\]

\[
(2) \quad Z = \frac{X-\mu}{\sigma}
\]

\[
(3) \quad P(a < X < b) = P\left( \frac{a-\mu}{\sigma} < Z < \frac{b-\mu}{\sigma} \right)
\]

According to the result of simulation, it is proved that our design for session state management is very reasonable with respect to hash collision rate and miss rate.

4 Implementation and Experiments

4.1 Implementation

Our SPI-based intrusion detection module was implemented on SGS prototype. Session State Manager of SGS is implemented on a Xilinx Vertex-II Pro XC2VP70 FPGA(7M Gate)[13] and Cypress CY7C1470V33 SRAM(72Mbit) using verilog HDL (Hardware Description Language) that is best suited for high-speed packet processing. The simulation of all functions were conducted by the ModelSim PE 6.1 simulator[14]. And all logics have been synthesized by Synplify Pro 8.4 tool[15].

In our prototype, main logics were implemented on three FPGA chipsets, Packet Processing Engine, Stateful Packet Inspection Engine, and Intrusion Detection Engine. Especially, the prototype we have developed focus on FPGA logic for real-time traffic analysis and SPI-based intrusion detection on high-speed links. Also, we employed inline mode capable of effective response by using four Gigabit Ethernet links as shown in Fig 6. The minimum clock period for data from input to output is 8ns which corresponds to a throughput of 2Gbps. That is, our system is capable of processing until a maximum throughput of full-duplex 2Gbps about incoming packets in FPGA Logic.

4.2 Experimental Results

If the SPI device is tracking TCP session state, then it has the potential to introduce denial of service when the session table becomes full(too many connections) or if it can’t keep up with the creation of new sessions(too many connections per second). That is, Max Concurrent Sessions(MCS) and Connections Per Second(CPS) are very important factors for performance evaluation.

We made use of the test bed shown in Fig. 9 for performance evaluation of our prototype system. The test bed consists of IXIA Traffic Generator[16], Gigabit Switch, Spirent Avalanche/ Reflector[18] and IDS Informer Attack Tool[17] for experiments.

In the results of measurement using Spirent Avalanche/Reflector, our system supported up to 40,000 Connections per Second.

Max Concurrent Sessions was measured as following procedure. First, a legitimate TCP session is opened through 3-way handshaking and then Spirent Avalanche
opens various numbers of TCP sessions from 500,000 to 1,500,000 with the Reflector. Exploit is transmitted which is required to trigger an alert in the initial TCP session. If the Session State Manager is still maintaining state on the first session established, the exploit will be detected. If the state table has been exhausted, the exploit string will be seen as a non-stateful attack, and will thus be ignored. Table 2 shows the results of alert generation in this experiment. As a result of experiment, we can see that our system supported up to 1,300,000 Max Concurrent Sessions.

![Fig. 7. Test bed for experiments](image)

Table 2. The Results of Max Concurrent Sessions

<table>
<thead>
<tr>
<th></th>
<th>First</th>
<th>Second</th>
<th>Third</th>
</tr>
</thead>
<tbody>
<tr>
<td>500,000</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>600,000</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>700,000</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>800,000</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>900,000</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>1,000,000</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>1,100,000</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>1,200,000</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>1,300,000</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>1,400,000</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>1,500,000</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

5 Conclusion

One of important requirements of SPI-based intrusion detection system is high performance. Even though SPI technology in network security system is developed to reduce false positive alerts, if not satisfied with performance, it may not be used. The performance of SPI-based intrusion detection system mainly depends on the performance of processing session table. In this paper, we proposed session state management scheme which can perform stateful packet inspection in real time by performing session table processing that allows more efficient generation of state information. And we designed and implemented SPI-based intrusion detection module in a FPGA to help alleviating a bottleneck in network intrusion detection systems.

Most of network security systems require capability to handle over one million concurrent sessions. Through the experiments, we proved that our proposed session state management scheme satisfies high performance.

References:

[7] Byoungkoo Kim, Youngjun Heo, and Jintae Oh, High-Performance Intrusion Detection in FPGA-based Reconfiguring Hardware, in Proceeding of APNOMS, 2005