IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 7, 2018, accepted June 11, 2018, date of publication June 25, 2018, date of current version July 19, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2850345

Improved Session Table Architecture for Denial of Stateful Firewall Attacks

ZOUHEIR TRABELSI[1], SAFAA ZEIDAN[1], KHALED SHUAIB[1], AND KHALED SALAH[2]
[1]College of Information Technology, United Arab Emirates University, Al Ain 15551, United Arab Emirates
[2]Department of Electrical and Computer Engineering, Khalifa University, Abu Dhabi 127788, United Arab Emirates

Corresponding author: Safaa Zeidan (safaa.z@uaeu.ac.ae)

This work was supported by UPAR under Grant 31T080.

ABSTRACT Stateful firewalls keep track of the state of network connections. The performance of stateful firewalls depends mainly on the processing of session tables and the mechanism used for packet filtering. This paper presents a stateful session table architecture for a splay tree firewall. A splay tree firewall organizes firewall rules in a designated prefix length splay tree data structure, combined with a collection of hash tables grouped by a prefix length. When using a splay tree firewall, packet filtering time is essentially reduced through multilevel filtering paths, where unwanted packets are rejected as early as possible. The proposed session table architecture reduces memory space consumption and packet filtering time, as it uses one hash slot per connection. Keeping information related to each connection in one session entry produces additional processing time, particularly for processing session timeouts. The proposed session architecture separates session state and timeout information into different data structures. Under DoS attacks, the proposed architecture compares non-first packets directly with a splay tree firewall. Consequently, packets are rejected early on, and thus avoiding the extra computational overhead caused by hash function calculation and session table processing.

INDEX TERMS Network firewalls, stateful firewall, session table, DoS attacks on session table, packet classification, early packet rejection, splay tree, hash table.

I. INTRODUCTION

Firewalls are the first line of defense against threats and attacks targeting private networks. The primary functionality of a firewall is to filter traffic routed in and out of a private network. This is done according to a predefined filtering policy rules, which are typically constructed to allow or deny a packet to pass through, depending on the packet's header information (protocol, source and destination IP addresses, source and destination ports).

First generation of firewalls, known as packet filter, perform packet filtering in a sequential order starting from the first rule until a matching rule is found. If no matching rule is found, the packet is processed by the default rule located at the bottom of the rule base. This type of firewalls does not store any previous connection related information, therefore named as stateless packet filter firewalls. In fact, each packet is treated as an individual entity, which is either routed if it matched the rule list or dropped if did not. Thus, filtering decision is made separately for every packet without taking into consideration earlier decisions for related packets. Stateless firewalls are fast and do not keep historical records, as they process incoming and outgoing packets independently and with no correlation of previously seen packets. On the other hand, for bi-directional services, such firewalls must have rules explicitly written to allow return traffic flows, which make them exposed to attacks, such as spoofing and fragmentation.

Nowadays, stateful or dynamic firewalls are the most widely used firewalls. They address limitations of stateless firewalls, as they keep track of all packets that belong to an existing flow from both directions using a session table. A session table, also named state table or connection track table, is part of the firewall internal structure. It tracks active connections and inspects packets to check whether it is part of a previous active session or not. If packets have the expected properties that the session table predicts, the packets are forwarded without any processing by the firewall filtering rules. Session table entries related information depends on the firewall vendor. But, they typically include <protocol (Prot), IP source (Src-IP), source Port (Src-P), IP destination (Dst-IP), destination Port (Dst-P), connection state and timeouts>, where connection

state is tracked until a connection is torn down or until a preconfigured timeout is reached. They may also include a sequence number, an acknowledgement number and a window size. The main advantage of stateful firewall is that it is more secure than stateless firewall, as the administrator no longer needs to write filtering rules to allow return traffic. This closes the security hole opened by rules that allow return traffic which can be exploited by an attacker to launch a denial of service attack (DoS). Furthermore, the rule list in stateful firewall becomes shorter as a single rule is enough to describe a flow; whereas two rules are needed for the same flow in the corresponding stateless firewall [1]–[3].

Likewise, application proxy firewalls are more secure than stateless firewalls. However, they are slower than stateful firewalls as application layer filtering is considered. When using a proxy firewall, two TCP connections are established on behalf of the packet source and destination. Thus, clients do not communicate directly with servers. The proxy firewall intercepts a request from a client, examines application payload then forward permitted packets to the destination server [3].

Apart from stateless firewall speed and rule conflicts problems, recent research tends to propose mechanisms to reject unwanted traffic as early as possible. Such type of traffic has a prolong effect on stateless firewall performance as it undergoes a long searching path until denied by the default rule. In our recent studies [4], [5], we proposed a splay tree firewall to address the aforementioned problems. It has been demonstrated that the splay tree firewall achieves high packet filtering speed, as it uses dynamic early multilevel packet filtering mechanism. In addition, the mechanism can be considered as a device protection technique against DoS attacks targeting the default policy rule. However, the model doesn't keep track of the states of established connections since it works as a stateless packet filter but not as a stateful firewall.

This paper proposes a firewall session table architecture based on an algorithm that can determine and maintain new, active and inactive session connections. Motivations and contributions in this paper fall into three main areas. **First**, current stateful firewalls keep all connection related information into a single session entry structure that basically consists of <Prot, Src-IP, Src-P, Dst-IP, Dst-P, state, timeout> [3], [27]–[31]. This structure produces additional processing time, particularly for session timeout processing [27], [28], [35]. To reduce costly processing of session timeout attribute, the proposed session architecture uses two different data structures for session entry information, one is used for session identifier <Prot, Src-IP, Src-P, Dst-IP, Dst-P>, while the other is used for session state and timeout <state, timeout>. **Second**, session tables such as the Connection track module in Netfilter [33], use two different hash entries per connection for the original and reply directions, whereas the proposed session table architecture uses one hash entry per connection. This will reduce memory space consumption, hash computation and filtering time used to identify if an arriving packet is part of an existing connection. **Third**, research related to stateful firewalls emphasizes on the session table architecture along with the packet filtering mechanism used, whereas most of the filtering process is done by comparing packet header information against the listed rules sequentially, resulting in an overall performance degradation [25]–[32], [34]. Existing literature either tackle session table architecture [27]–[32], or provide algorithms for fast packet filtering the naïve one [4], [5], [16]–[26]. However, to the best of our knowledge there is no prior work that integrates enhanced session table architecture with fast packet filtering mechanism. Thus we propose an enhanced session table architecture and integrate it with a fast packet filtering mechanism (splay tree firewall [4], [5], as well as study the overall system behavior under normal and attack situations. Usually, under firewall normal operation all session table entries represent valid flows. However, some types of abnormal activities can fill up the session table with invalid entries, raising security and performance concerns. In such situations, the proposed algorithm switches to normal filtering using a dynamic splay tree mechanism with its early packet rejection characteristic. The main contributions of this paper can be summarized as follows:

- Propose an enhanced session table architecture that:
  - Separates session entry into two different data structures to improve session lookup and enhance timeout processing (section 5.1 and 5.2).
  - Uses one hash slot per connection to save memory space, reduce hash computation, and increases firewall concurrent connections (section 5.1).
  - Expands the hash table vertically by reducing the number of buckets using the same memory space to resolve collision (section 5.3).
  - Reduces the effect of a DoS attack (section 5.6).
- Propose a complete statefulness solution by integrating the proposed session architecture with splay tree packet filtering mechanism (section 5.5).
- Study the behavior of the overall system under normal and attack situations (section 5.6), considering:
  - Attacks such as first packet and high percentage of non-matching traffic attacks can be mitigated using the nature structure of the proposed session table architecture or the splay tree firewall without any additional overhead (section 5.6 A and B).
  - Other non-first packet attacks can be mitigated by first verifying the splay tree firewall (section 5.6 C).

The paper is organized as follows. Section II describes the state of the art of stateful firewalls. Section III provides details about splay tree firewalls. As an example, section IV discuses factors affecting session table processing time in a connection tracking system. Section V describes the proposed session table architecture. Implementation and experimental evaluation is demonstrated in section VI. Finally, Section VII concludes the paper with some insights and related perspectives.

II. RELATED WORK

Previous research work related to functional enhancement of stateful firewalls can be categorized into four different areas, and most of them provide straight forward adaptations of methodology and techniques previously designed for stateless firewalls.

A. RULE ANOMALY DETECTION AND ELIMINATION

The first area focuses on stateful firewalls rules anomaly detection to discover and release conflict, redundant and shadowed filtering rules. This can help in reducing policy size, memory space consumption and packet filtering time on a firewall. However, most previous research methods such as [6]–[9] were limited to solve anomalies in the configuration of stateless firewalls, whereas little work has been done for stateful case.

In [10], an approach is proposed to describe stateful firewall model which split into stateful and stateless components. In [11], a solution is proposed based on adapting the existing anomaly detections techniques for stateless firewalls to suit stateful firewalls, but without considering session table management anomalies. In [12], an approach is proposed based on semantic web technologies to model both stateless and stateful firewalls. Cuppens *et al.* [13] provided algorithmic solutions based on specifications of general automata to solve flawed configurations, whereas in [14] they completed their algorithmic solutions to cover intra-state rule misconfiguration for stateful rules and inter-state rule misconfiguration for both stateful and stateless rules. Basile and Lioy [15] extended the Liu's model of stateful firewalls [10] to the application level, in order to detect rule pair and multi-rule anomalies, hence reducing the likelihood of conflicting and suboptimal configurations.

B. USE OF HIGH PERFORMANCE HARDWARE

The second area discuses high performance hardware architectures such as using FPGA for speed performance and minimizing memory requirements. Most research work proposed in this area focused generally on packet classification problem. In [16], a decision tree and heuristic rule set partitioning algorithm is used to reduce rule replication and memory requirement. Similarly, in [17] a rule categorization algorithm by considering rules field features is proposed based on parallel and pipelined architectures on FPGA to minimize memory requirements. In [18], a massively parallel firewall circuit is presented to develop customized firewall circuits in the form of synthesizable VHDL code for FPGA configuration. However, FPGA based solutions require complicated processes to be done on the FPGA circuits in case of data structure or rules change.

C. USE OF ADVANCED PACKET FILTERING TECHNIQUES

Research on the third area focuses on proposing filtering techniques to enhance the search speed of stateful firewalls, through inheriting previous research techniques proposed for stateless firewalls adapted with session table management. Previous work to enhance the filtering optimization problem utilized one or more of the followings: traffic awareness techniques, early packet rejection and acceptance techniques, and reordering of dependent and independent filtering rules as well as rule-fields.

1) TRAFFIC AWARENESS TECHNIQUES

Researches presented in [19]–[21], were the first to have a special focus on the network traffic statistics and utilize the traffic characteristics in improving the average filtering time. In [19]–[20] an improvement of alphabetic tree is proposed using traffic statistics. However, the statistical trees were not able to scale well with the number of fields values. To solve this problem, El-Atawy *et al.* in [21] proposed a technique that is based on statistics collected from policy segments in order to build Huffman trees that dynamically adapt to the traffic statistics.

2) EARLY PACKET REJECTION AND ACCEPTANCE TECHNIQUES

The idea of firewall optimization through early packet rejection and acceptance was introduced in [19], [20], and [22]. Hamed *et al.* [19], [20] were the first to exploit the idea of early rejection of unwanted traffic flows without impacting other flows. A technique named field value set cover (FVSC) built a number of rejection rules that were examined before the real firewall policy. However, this technique suited only for small security policies with low diversity of field values. Thus, in [22] Policy Boolean expression relaxation (PBER) technique is proposed using binary decision diagrams (BDD) to implement the Boolean expression of the policy acceptance space.

3) REORDERING OF RULES AND RULES FIELDS

Wang *et al.* [23], [24] and Trabelsi *et al.* [25], [26] addressed the case of firewall packet optimization for independent filtering rules. In [23], an optimizing algorithm based on Markov model was proposed. The algorithm rebuilt the rule list by using firewall decision tree algorithm. However, as a result of using Markov model, the rule order changed for every incoming packet, consequently reducing the firewall efficiency. To overcome this problem, authors in [24] improved the firewall ordering algorithm in [23] by using a statistical model in which there is no need for rule order adjustment for every incoming packet. In [25] and [26], we were the first to propose and evaluate a mechanism based on rules and rule-fields reordering and focus on its major effect in reducing the overall packet processing time. In [25], the rule/rule-fields reordering processes were done at the end of each traffic window which may produce an overhead proportional to the number of filtering rules and rule-fields. Hence, in [26] a disjoint system stability test was performed, one for the filtering rules and the other for each rule-field. Also, the error precision was used and the efficient traffic window size effect was intensively studied.

Our recent study in [5] encompasses all the aforementioned sub categories, where a new type of firewall called splay tree firewall is proposed that can handle early packet rejection and acceptance, and can perform splay filters reordering based on a statistical model that utilizes traffic characteristic. The paper discuses an efficient solution that takes into consideration the position of each splay filter with respect to the previous ones. This allows best prediction of the order of the filters, and consequently optimizes the filtering time of the next network traffic window. Finally, the paper offers more understanding in terms of splay filters order combinations and their effect in the total packet filtering time and on how to choose the best initial order to start with. However, the mechanism does not keep track of connection states.

D. SESSION TABLE PERFORMANCE

The fourth area concentrates on session table architecture and performance to enhance session entry creation, lookup and deletion processes. In [27], a fast session table manipulation algorithm is proposed through enhancing the PATRICIA trie, that is employed in some session tables, in order to improve timeout process. Li *et al.* [28] generalize their proposed algorithm in [27] to cover multi-queue architectures and defend hosts against SYN flood attack. In [29], a connection tracking system for tree rule firewall is proposed to reduce memory space consumption and processing time. However, Chomsiri *et al.* [30] claim that hashing computation used in stateful tree rule firewall [29] takes more time than comparing packet headers with respective conditions specified in the corresponding firewall rule. Therefore, they proposed a hybrid mechanism [30], in which stateful and stateless rules were mixed together. Despite that the resulted rule set is conflict free, the number of rules exponentially increased with the diversity of rule fields values, which make this technique not suitable for security policies with large numbers of rules. In [31], a design and implementation of session management architecture in FPGA is proposed; whereas Xu *et al.* [32] presented an architecture aware session lookup scheme for deep inspection on network processors.

However, in previous session table related work, no clarification was given on the used packet filtering mechanism and its integration with the proposed session table architecture. Moreover, no solutions were proposed for session timeout processing enhancement, number of slots used per connection and the capability of the proposed session architecture in thwarting DoS attacks.

To the best of our knowledge, all research work done in the field of stateful firewall performance optimization focused on packet filtering mechanisms and session table architectures. However, no complete optimization mechanisms based on the integration of both packet filtering and session table architecture had been proposed to date in the literature. The demand for a complete integration of these two parts and the behavior of session table under normal and attack situation motivates this research.

In the next section, we give a brief background on splay tree firewall (as packet filtering mechanism used in this paper), highlighting its advantages and how to utilize such a search technique in reducing substantially packet filtering time. Then, we take it a step further to propose an enhanced session table architecture to achieve statefulness characteristic.

III. STATELESS SPLAY TREE FIREWALL

A traditional firewall filters network packets by comparing packet headers sequentially against a set of predefined filtering rules. Each filtering rule is defined by a set of filtering fields, and associated with an action to either block or forward a packet to its destination. A significant drawback of the linear search is that, unwanted traffic targeting specific rules such as the default filtering rule may cause more harm than others by producing an overhead to the system. This overhead is proportional to the number of rules and fields used in the security policy. Such unwanted network traffic may cause a DoS attack situation and consequently may degrade considerably the firewall performance [4], [5], [19]–[22]. Thus, it is very important to reject such network traffic as early as possible.

In [4] and [5], we proposed a splay Tree firewall to optimize early acceptance as well as early rejection packet filtering paths. This firewall achieves a high processing speed as the splay tree data structure changes dynamically according to traffic flows. However, the proposed design works only as a packet filter with empowered filtering enhancement, treats packets individually and does not keep track of network connections states.

Splay Tree firewall consists of several *splay filters* namely, protocol, source/destination addresses, source/destination ports, etc. More attribute filters can be added, deleted or modified easily. Each splay filter consists of a splay tree data structure and a collection of hash tables. The splay tree data structure adapts dynamically according to the network traffic flows allowing early acceptance for repeated packets (optimization in the acceptance path). On the other hand, splay tree node with minimum prefix length (min-Node) is maintained at root→left position to reject unwanted packets by at most two memory accesses (optimization in the rejection path). Fig.1 shows a splay tree firewall with three arbitrary ordered splay filters named F1, F2 and F0.

All splay tree filters are connected in a consecutive order using multi-filtering levels in which packets are rejected as early as possible. However, the order of these filters is dynamically determined by their packet rejection rate. This reduces the time required for comparing packets with splay tree filters as well as early rejecting packets that do not match any splay tree filter. Packets will not propagate to the next adjacent splay tree filter until they pass a cascaded filtering level. The firewall will continue filtering packets using certain order of splay tree filters under a certain threshold qualification (Chi-Square stability test) to compromise between performance and cost overhead. In case of a DoS attack, the order of the

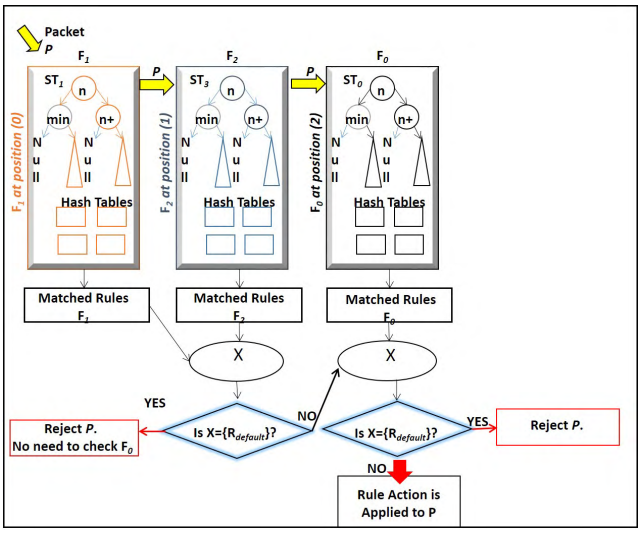

FIGURE 1. Splay Tree Firewall with three arbitrary filters.

splay filters may change according to the Chi-Square stability test decision, which is based on the DoS attack significance.

The splay tree firewall uses light weight traffic adaptive mechanism to filter unwanted expensive traffic and significantly minimizes packet filtering time with reasonable memory space requirement; thus saving the firewall processing power and increasing the overall throughput. Also, the overhead of reordering the splay tree filters according to traffic characteristic is guaranteed to be lower than the gain achieved in filtering time.

However, the splay tree firewall is basically a stateless packet filter in which filtering decision is made separately for every packet and does not keep history for earlier decisions made on related packets. Although the splay tree firewall with its dynamic structure shows resiliency against complex attacks targeting bottom rules [34], it may still suffer from some DoS attack types due to the security hole opened by rules allowing return traffic (response traffic).

To inherently provide more security for the splay tree firewall and eliminate the aforementioned limitations of stateless firewall, we propose a session table architecture that can record and process different connections' states to protect the network. Connection tracking system within the Netfilter Model is considered as an example of an open source session table. From the discussion, we highlight connection tracking system structural limitations and derive factors that affect session table performance.

IV. CONNECTION TRACKING SYSTEM

Generally, the performance of stateful firewalls depends primarily on the performance of session table processing and the packet filtering mechanism used in the security policy [27]–[29], [31], [46]. The session table basically maintains entries for active connections in a memory structure. Each entry stores the most important information of streams that are allowed by the firewall rules' database.

Prot	Src_IP	Dst_IP	Src_P	Dst_P	State	Timeout

FIGURE 2. Session table entry general format.

Such information may include: Prot, Src-IP, Dst-IP, Src-P, Dst-P, State, Timeout and other characteristics. Fig. 2 shows the session table entry general format.

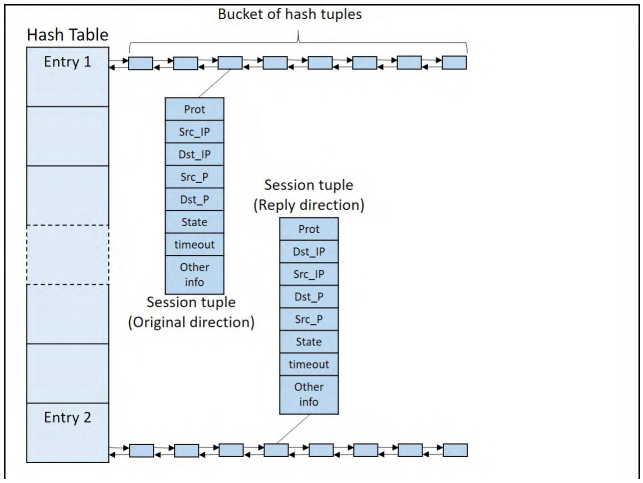

FIGURE 3. Connection tracking system session table structure.

Connection tracking system within the Netfilter, is an example of stateful mechanism modular that is used by IPtables [33]. As shown in Fig. 3, this system consists of a hash table where each hash table entry has a bucket of linked list of hash tuples. The maximum bucket length is 8. Packets information are hashed to these tuples using a hash function (Jenkins' hash). It is important to mention that for each connection there are two hash tuples in different buckets. One tuple to store request packet's information for the original direction from source to destination, and the other to store reply packet's information for the reply direction from destination to source.

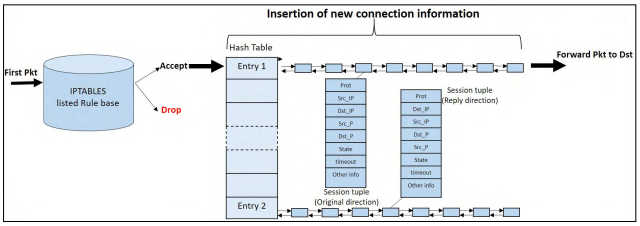

FIGURE 4. Connection tracking system lookup process for first packet.

Connection tracking system lookup process illustrated in Fig. 4, verifies the first packet of a connection against the firewall rule database. If it matches a filtering rule with accept action, then an entry is added to the session table, otherwise the packet is dropped. However, non-first packets shown in Fig. 5, will be verified directly against the session hash table and buckets, to check if they are part of an active

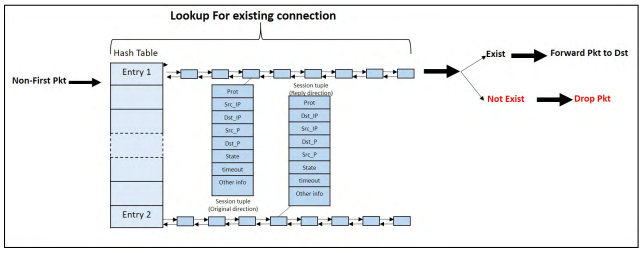

FIGURE 5. Connection tracking system lookup process for non-first packet.

connection or not. If the non-first packet matches an entry in the hash table and buckets, it will be forwarded to its destination without checking the rule set. Otherwise, the packet is dropped.

Thus, if a first packet of a connection is received, it will be verified by the IPtables rules. If the packet matches a filtering rule, a hash function is applied to the five attributes of the packet header to obtain hash entry number (entry 1), as shown in Fig. 3. This entry number will indicate in which bucket the original connection information will be stored. Also, the hash function is applied again to obtain a hash entry number (entry 2), Fig. 3, to indicate the bucket in which reply attributes to be stored. Similarly, terminating a connection needs two operations to delete the corresponding connection's tuples from memory. Entry 1 and Entry 2 are calculated as follows:

Entry 1 = Hash(Prot, Src_IP, Dst_IP, Src_P, Dst_P, seed)

Entry 2 = Hash(Prot, Dst_IP, Src_IP, Dst_P, Src_P, seed)

Where *seed* is a random number generated by the IPtables.

A. STRUCTURAL FACTORS AFFECTING SESSION TABLE PERFORMANCE

The aforementioned connection tracking system structure raises memory and time limitation issues, that influence the session table performance significantly. The structural factors affecting session table performance are as follows:

1) NUMBER OF NODES PER CONNECTION

Two hash tuples are used, one to store request information for the origin direction and the other to store reply information for the reply direction.

2) MEMORY SPACE

Entries 1 and 2 (Fig. 3) representing the same connection, invoke the hash function twice in case of connection tuples insertion and deletion as well as it has to allocate or remove memory two times for the two tuples. In this case, Netfilter model is not able to use memory space efficiently which reduces the maximum number of concurrent connections due to the use of pair tuples for each connection.

3) HASH FUNCTION USAGE

The frequent use of the hash function itself, twice per connection insertion and twice per connection removal, is time consuming when dealing with a large number of network packets requesting or terminating connections.

4) SESSION TIMEOUT PROCESSING

Another important factor is session timeout processing. Existing stateful firewalls have high timeout processing [27]. This is due to the fact that all connection's information is stored in a single session entry, as shown in Fig. 2. Thus, arriving packets are looked up in the session table for a match. If a match is found indicating that packets are related to an existing connection, then session state and timeout will be updated. In normal condition, these match-update operations will be done frequently. Moreover, in case of a DoS attack, such as TCP SYN flooding, excessive invalid entries are added to the session table from spoofed Src-IP, causing session table to be filled. Therefore, this would rise security and performance concerns. On the other hand, when a connection is terminated or over timed, its entry will be deleted from the session table to eliminate inactive sessions and therefore reducing security holes. Hence, timeout processing also plays a vital role in session table performance.

V. PROPOSED SESSION TABLE ARCHITECTURE

Considering the aforementioned factors and limitations, we propose a new session table architecture based on session entry attributes separation, as shown in Fig. 6. The new session table architecture involves hash table, uses one session node per connection to consume less memory and operates in a faster manner under normal and attack conditions.

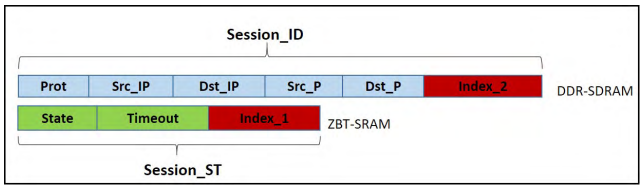

FIGURE 6. Proposed session table entry format.

Instead of using one session entry containing all connection information, we will use two different memory structures, DDR SDRAM and ZBT SRAM, to store relative connection information providing only one session node per connection. Since the session entry shown in Fig. 2 is wider than 16 bytes and concurrent connections may reach up to 1 million entries, over 128 MB of memory space is required to store relative session table. Therefore, the use of DDR SDRAM and ZBT SRAM can greatly enhance the session table performance. The proposed session table entry format in Fig. 6 consists of the structures, Session_ID and Session_ST pointing to each other using Index_1 and Index_2.

A. SESSION_ID STRUCTURE

High speed DDR SDRAM memory will be used to store the five connection's attributes < Prot, Src_IP, Dst_IP, Src_P,

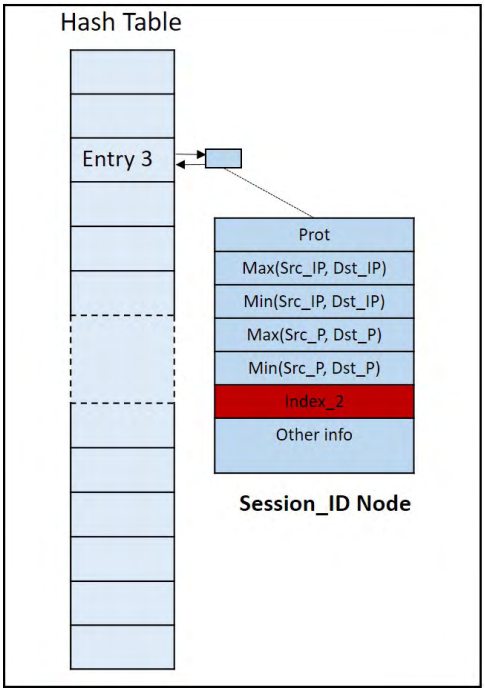

FIGURE 7. Proposed Session_ID node structure.

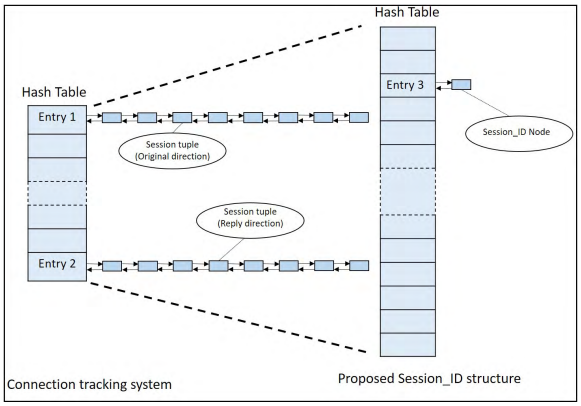

FIGURE 8. Proposed Session_ID structure vs Connection tracking system.

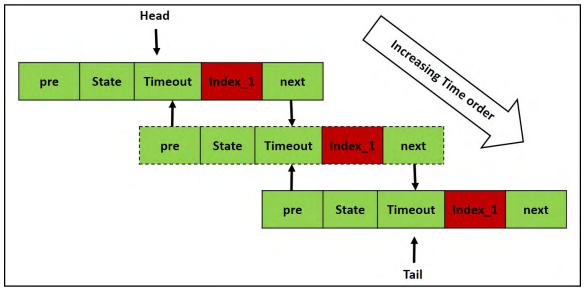

FIGURE 9. Proposed Session_ST structure.

Dst_P> augmented with Index_2, in a unique Session_ID, where Index_2 will lead to the connection <state, timeout> attributes. Only one node per connection is used to optimize session table memory usage. Fig. 7 illustrates Session_ID node structure, in which packet attributes < Prot, Src-IP, Dst-IP, Src-P, Dst-P> are hashed using a hash function to obtain entry 3. This single entry indicates the bucket in which Session_ID is stored. Entry 3 is calculated using the following function:

Entry 3 = Hash(Prot, max(Src_IP, Dst_IP), min(Src_IP, Dst_IP) , max(Src_P, Dst_P), min(Src_P, Dst_P), seed).

Jenkins hash [51] is used in the proposed session table as it provides fast operations and is capable of distributing its 32 output bits randomly.

In the proposed architecture, a conneciton is represented by a single node; whereby two nodes are used in the Netfilter model, one for request and the other for reply. Thus, memory space will be saved by 50%, which implies that the number of concurrent connections can double than that in the Netfilter model, Fig. 8.

Additionally, the proposed Session_ID is identified by a single entry (entry 3); while two entries are used in Netfilter model (Entries 1 and 2). Therefore, calculation and hashing for session entry insertion or deletion are done twice for the same connection in the Netfilter model, while hash function is invoked only once per connection in the proposed architecture. This can reduce session table processing time significantly, and therefore enhances the overall firewall performance greatly, Fig. 8.

B. SESSION_ST STRUCTURE

Remaining session entry attributes <state, timeout> augmented with Index_1 are stored in Session_ST in ZBT SRAM using double linked list as shown in Fig. 9. Where Index_1 will lead to the connection's Sesion_ID.The state attribute stores corresponding connection current state information. Whereas, timeout attribute is used to determine which session entry has to be removed if the session table is full. The timeout attribute is updated by an internal timer object each time the corresponding Session_ID is accessed. Differently from the existing stateful firewalls, the proposed Session_ST structure will avoid reading many entries when processing connections' timeout attribute. This is because Session_ST is stored in increasing order of time. This is very important as if any session entry timeout doesn't expire, all subsequent connections' entries, that follow after this entry, are also valid and do not timeout too. Thus, in case that session table is full and new sessions cannot be allocated, instead of comparing the time of current internal timer with timeout of each session entry, as in existing stateful firewalls, the time of current internal timer will be compared to the timeout of Session_ST node at the head of the double linked list only. Therefore, unused open sessions that consume firewall resources will be easily identified and removed from the session table. This structure also will help to protect the firewall against DoS attacks targeting its session table, since old/unused/excessive

sessions will reside close to the head of the double linked list, according to their timeout. Therefore, they can be defined and removed easily enhancing firewall performance and allowing it to accept newly connection requests.

It is important to mention that session timeout attribute should be managed separately for embryonic and established connections. Embryonic state connection, known also as half open connection, is a connection request with unfinished TCP handshaking; while established state connection has completed successfully the handshaking. Therefore, embryonic connections need to have shorter timeout compared to established connections to prevent the firewall from DoS attacks, such as TCP SYN flooding attack. Kim *et al.* [35] proved that memory allocation for embryonic TCP connection should not exceed 10 seconds of inactivity.

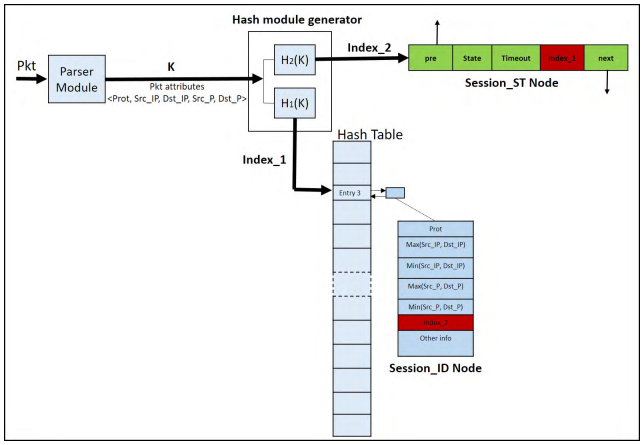

FIGURE 10. Proposed session table architecture.

C. SESSION TABLE ARCHITECTURE COMBINING SESSION_ID AND SESSION_ST NODES

Session_ID and Session_ST memory blocks are related to each other using Index_1 and Index_2 addresses generated by hash module generator as shown in Fig. 10. The hash module generates indices and manages session entries based on the five attributes extracted from the received packet using packet parser module. Two hash functions are used in the hash module generator, H1(k) and H2(k). H1(k) is used to generates hash indices pointing to Session_ID located in DDR-SDRAM memory and to solve for collision in order to achieve faster session table lookup. H1(k) is embedded with min() and max() functions to generate a single node per connection. The other hash function H2(k), is used to generate addresses for corresponding Session_ST in the double linked list in ZBT SRAM to enhance session table timeout processing. The proposed session table architecture expands the DDR SDRAM hash table vertically by reducing the maximum bucket length. This method will decrease hash collision using the same memory space compared to Netfilter model, as more hash slots will be available for new connection requests. For instance, if the bucket size is reduced to half, the hash table size will be doubled using the same provided DDR SDRAM memory space, which implies that collision can be decreased by approximately 50%. Typically, one node per bucket is used in the proposed session table architecture resulting in a hash table that is 8 times bigger than that in the Netfilter model.

D. PROPOSED SESSION TABLE OPERATIONS AND ALGORITHMS

There are 4 operations associated with session table processing: (1) insert a new session entry (Session_ID, Session_ST) when first packet's acceptable request is received, (2) lookup the session table for a matching Session_ID entry when non-first packet is received, (3) update a Session_ST state and timeout attribute, and (4) delete a session entry (Session_ID, Session_ST) either when termination packet is received or Session_ST is over timed. Lookup and update operations are usually related to each other. When non-first packets belonging to an existing session match a session table entry, session state and timeout attributes will be updated.

1) SESSION ENTRY INSERTION ALGORITHM

When first packet request is received, it is verified by the splay tree firewall. If the packet does not match a rule, it will be rejected as early as possible via splay tree minNode. However, if the packet is accepted by a rule, then hash functions H1(k) and H2(k) are applied to the packets' five attributes to get Index_1 and Index_2. DDR-SDRAM memory block is filled with Session_ID attributes, augmented with ZBT SRAM memory block address pointed by Index_2. Session_ID node is inserted in the hash table. Also, ZBT SRAM memory block is filled with Session_ST attributes augmented with DDR-SDRAM memory block address pointed by Index_1. Session_ST is inserted in the double linked list and appended to tail.

2) SESSION TABLE LOOKUP AND ENTRY UPDATE ALGORITHM

When non-first packet is received, hash function H1(k) is applied to the packets' five attributes. The hash table is searched for matching Session_ID. If a matching is found, Index_2 is used to reach corresponding Session_ST node. Then, Session_ST attributes are updated with the new state and current time. After that, Session_ST node position is shifted to the tail of the double linked list.

3) SESSION ENTRY DELETION ALGORITHM

Defining the beginning and the end of a connection depends mainly on the protocol used. For connection oriented protocols, such as TCP, the firewall can explicitly identify packets requesting the beginning and the end of a session. However, for connectionless protocols like UDP the deletion of a session entry is determined by inactivity timeout. Since Session_ST is stored in increasing order of time, it will be easily to determine and remove timed out Session_ST and corresponding Session_ID node from the session table.

If a time interrupt is received, the first Session_ST node pointed by head in the double linked list is read. The current time is compared to Session_ST timeout attribute. If they are not equal, exit the process and no need to search for other session entries that may over timed. However, if they are equal indicating that this Session_ST node is over timed, then Session_ST node is deleted, and the head of the double linked list is updated. The corresponding Sesion_ID is also deleted using Index_1.

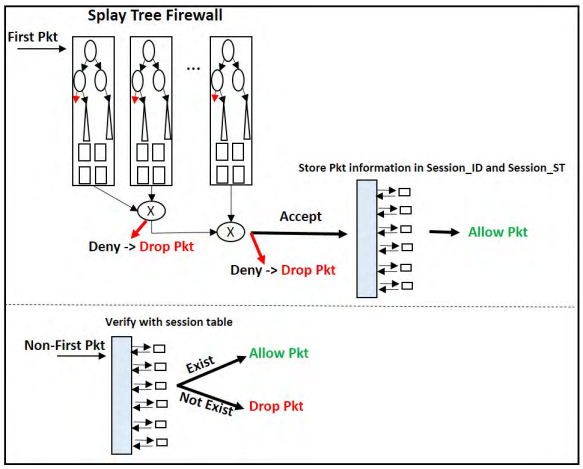

FIGURE 11. Stateful splay tree firewall first and non-first packet processes.

E. FIREWALL SYSTEM INTEGRATION

Splay tree firewall is integrated into the proposed session architecture to provide a complete statefulness solution towards fast packets acceptance or rejection with enhanced session management operations. First and non-first TCP packets shown in Fig. 11 are used to illustrate packet life cycle and operations in the overall integrated system.

Upon TCP packet arrival, if this is a first packet requesting connection (SYN), then splay tree filters are searched for matching as follows [5]: the first statistical splay filter is searched using binary search on prefix length. If the packet doesn't match the first statistical splay filter, it will be rejected early by its min-Node. Otherwise, the search continues to the second statistical splay filter. If the packet matches the second filter, then the list of matching rules will be intersected with the list of matching rules from the previous filter (Cascaded filtering level). If the intersection is empty, the packet will be rejected early. Otherwise, the search continuous to the next statistical filter, and so on. Statistics during the above stages will be collected to be used later in reordering the splay filters according to their rejection rate.

At this stage, the packet represents a valid flow requesting a connection. Then, H1(K) and H2(k) generate Index_1 and Index_2, where Session_ID and Session_ST are hashed to their corresponding data structures using session entry insertion algorithm. Finally, the packet is forwarded to its destination.

However, if the TCP packet is a non-first packet (ACK), then the session table is searched for a matching Session_ID using session table lookup and entry update algorithm. In which, if a matching is found, corresponding Session_ST node attributes are updated and the node is appended to the tail of the double linked list. Therefore, the packet is forwarded to its destination. Otherwise, the packet is dropped.

Likewise, if a termination packet is received (FIN, RST), then the session table is searched for a matching Session_ID using session table lookup and entry update algorithm. In which, if matching is found, Session_ID and corresponding Session_ST will be removed using session entry deletion algorithm. However, if the connection is idle (no packet is received) and timeout is reached, it will be more easy to identify such connections, as they will reside close to the head of the double linked list. Therefore, Session_ST and corresponding Session_ID will be removed from the session table using session deletion algorithm.

F. PROPOSED SESSION TABLE ARCHITECTURE FOR DEFENDING AGAINST DOS ATTACKS

To date, stateful firewalls suffer from slow rate distributed DoS attacks (DDoS), that can wreak havoc and exhaust all of the firewall's resources, especially the session table [36], [37], [47]. Most of stateful firewall vendors mitigate DoS attacks using different mechanisms such as threshold based mechanisms and SYN cookies, as in Screen features in Juniper Networks [38]. For instance, to save a firewall session capacity, the total number of sessions that can be established to/from a specific IP address is limited via thresholds. However, if threshold activation is set high, then attack traffic will pass through the firewall and can consume its resources. Moreover, most of these threshold based mechanisms are often not enabled and even if enabled, they dramatically increase CPU and session table utilization, such as in TCP SYN flood mechanism [39].

In this paper, DoS attacks are mitigated in the integrated system using three different methodologies. The first methodology makes use of the adaptive nature of the splay tree data structure, where no complex session table operations and hashing are involved; saving the firewall's resources and improving packet filtering time. Additionally, no threshold mechanisms are used that would produce an overhead to the system. In the second methodology where session table is involved, DoS attacks are mitigated using the novel architecture of the proposed session table. The third methodology defends the system against non-first packet attacks targeting to keep the firewall busy by doing expensive hash calculation and session table operations. In such a situation, the system will consult splay tree filters in order to drop the attack packets as early as possible.

1) MITIGATING ATTACKS WITHOUT SESSION TABLE INTERVENTION

The nature of the splay tree structure is based on using the min-Node that allows for fast packet rejection, and the

locality of reference that allows for fast packet acceptance for skewed network traffic. This locality of reference is based on the notiton that if a node in the splay tree is accessed, then it is likely to be accessed again in the near future [5].

Splay tree maintains the min-Node at root→left position. Any attack traffic with high percentage of non-matching packets will be dropped earlier via splay filter min-Node, by at most two memory accesses (root→min-Node→Null). Even if such attack with high percentage of non-matching traffic is designed to target a specific splay filter with high position, to increase the filtering time, it will also be rejected earlier. The reason is that splay filters are also reordered according to their min-Node and cascaded rejection rate. This reordering will adjust the targeted splay filter position to reside at lower ones, and consequently the attack is rejected earlier by its min-Node.

Furthermore, in crafted attacks designed to flood a specific splay filter via huge matching traffic, the benefits of splay tree locality of reference are used. However, the cascaded level at the next splay filter will prevent such attacks from propagating deep in the consequent splay filters [5].

Let $SF_{p,w}(i)$ represents splay tree filter i at position p in a certain traffic window w, where i $\epsilon\{1, 2, \ldots, I\}$ represents tuples used in the firewall <Prot, Src-IP, Src-P, Dst-IP, Dst-P>, respectively. Where I in this case equal to five. If packet t is rejected by $SF_{p,w}(i)$ min-Node, then its min-Node counter represented by $m_{SF_{p,w}(i)}$ is incremented by "1", that is:

$$m_{SF_{p,w}(i)} = m_{SF_{p,w}(i)} + 1 \tag{1}$$

where the initial state of $m_{SF_{p,w}(i)}$at the beginning of each window w is "0".

Likewise, if packet t is rejected by $SF_{p,w}(i)$ cascaded level, then its cascaded counter represented by $x_{SF_{p,w}(i)}$ is incremented by "1", that is:

$$x_{SF_{p,w}(i)} = x_{SF_{p,w}(i)} + 1 \tag{2}$$

where the initial state of $x_{SF_{p,w}(i)}$ at the beginning of each window w is "0".

At the end of each network traffic window, the amount of packets entered, $t_{in}(p)$, and left, $t_{out}(p)$, each splay filter $SF_{p,w}(i)$ at position p, can be defined in terms of min-Node and cascaded rejection as follows:

$$t_{in}(p) = \begin{cases} T, & p = 1 \\ t_{in}(p-1) - [m(p-1) + x(p-1), & 1 < p \leq I \end{cases} \tag{3}$$

$$t_{out}(p) = \begin{cases} T - m(p), & p = 1 \\ t_{in}(p-1) - [m(p-1) + x(p-1), & 1 < p \leq I \end{cases} \tag{4}$$

Where T represents the total number of filtering packets in window w.

Using $t_{in}(p)$ and $t_{out}(p)$ defined in Eq. 3 and 4, the probability of splay filter $SF_{p,w}(i)$ to reject a packet can be defined as follows:

$$P_w(p) = \frac{t_{in}(p) - t_{out}(p)}{t_{in}(p)}, \quad 1 < p \leq I \tag{5}$$

To check whether or not there is a need to reorder the splay filters, a Chi-square test of homogeneity is carried on previous and current windows $(w-1)^{\text{th}}$ and w^{th}, respectively as follows:

$$\chi^2 = \sum_{w-1}^{w} \sum_{p=1}^{I} \frac{(m(p) - e\_m(p))^2}{e\_m(p)} + \frac{(x(p) - e\_x(p))^2}{e\_x(p)} + \sum_{w-1}^{w} \frac{(t_{out}(I) - e\_t_{out}(I))^2}{t_{out}(I)} \tag{6}$$

Where the symbol e represents the expected value of the corresponding variable, refer to [40] for expected value calculations. Splay filter are reordered in descending order according to their rejection rate, Eq. (5), if indicated by the chi-square test.

At this level, attacks are prevented without any intervention of threshold mechanisms or session table operations and hashing. As a result, the filtering time can be saved and the system throughput is enhanced.

2) MITIGATING ATTACKS WITH SESSION TABLE INTERVENTION

The difference in session timeout for initial and established sessions allows the firewall to remove entries generated by first packet attacks, which depend mostly on spoofed src_IP with no expected reply, such as TCP SYN Flood, while maintaining legitimate entries. The entries removal process can even get enhanced a lot using the proposed session table architecture, as it separates Session_ID and Session_ST into two different data structures. This separation will help in solving the problem of highly cost timeout attribute processing found in most stateful firewalls [27]. Sessions that timed out are easy to identify and remove from Session_ST, as they reside close to the head of the double linked list. If first packet attack targets to fill the proposed session architecture with excessive sessions, then old Session_STs close to the head of the double linked list will be rapidly removed, using session entry deletion algorithm. Since these sessions were inserted in increasing time order, Fig. 9. By this, the firewall can continue accepting newly connection request.

3) MITIGATING NON-FIRST PACKET ATTACKS BY VERIFYING THE SPLAY TREE FILTERS FIRST

In the two previous attack prevention methodologies, attacks are mitigated using the nature of the splay tree firewall or the proposed session architecture. However, there are other types of attacks that are designed to make the firewall intensively performs expensive hash calculations and session lookups. These attacks are defined as non-first packet attacks, such as Reset flood attack [41] and Ack flood attack [42]. For instance, in an ACK flood attack, attackers send spoofed ACK packets at very high rates that do not match any session

entry within the firewall session table. Therefore, exhausting the firewall by forcing it to perform intensive session table lookups. Since such attacks are considered as non-first packets, they will be verified against the session table using hashing and session table operations, as shown in Fig. 11. This would result in performance degradation. Therefore, if such attack packets are verified by splay tree filters and are rejected, then hashing calculation time can be saved. This is based on the fact that if a non-first packet is denied by splay tree filters, then there is no need to further check the session table; since no chance that this packet information will have an entry in the hash table.

This methodology depends mainly on monitoring the network traffic to understand its normal operating behavior and establish baseline measurements to detect abnormal activities. Therefore, any changes in the baseline measurements due to abnormal or burst non-first packet traffic will indicate that the system is under DoS attack. Hence, the proposed system will verify the traffic firstly against splay tree firewall instead of the session table.

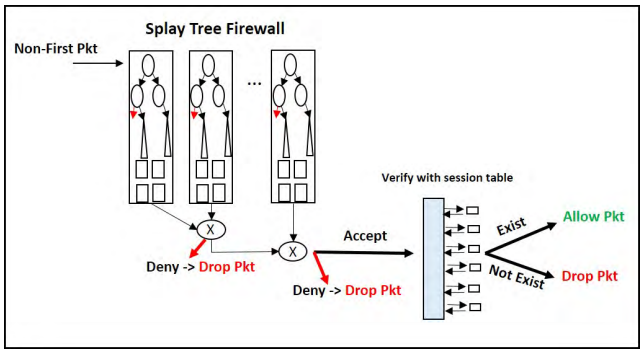

FIGURE 12. System behaviour in case of non-first packet attack.

Here, the system should distinguish between attack and legitimate packets. As shown in Fig. 12, if packets do not match the splay tree firewall, then this attack is mitigated without session table intervention, section A. However, if packets match the splay filters, then packets should be verified by session table to check if they belong to any current session or not. If match is not found in the session table, then this attack is mitigated with session table intervention, section B. However, if entry is found in the session table indicating legitimate traffic, then packets are forwarded to its destination.

Adaptive threshold algorithm [43] is used in this methodology to measure ACK packets and compare it with previously defined threshold. Let a_t represents ACK packets arrival rate at time t. The mean of a_t is computed using Exponential Weighted Moving Average (EWMA) as follows:

$$\mu_t = \beta\mu_{t-1} + (1-\beta)a_t \tag{7}$$

Where β is the EWMA factor.

If $a_t \geq (\propto +1)_{\mu_{t-1}}$ for $\propto \geq 0$, then the system will switch to splay filters to filter the packets firstly instead of the session table as indicated in Fig. 12.

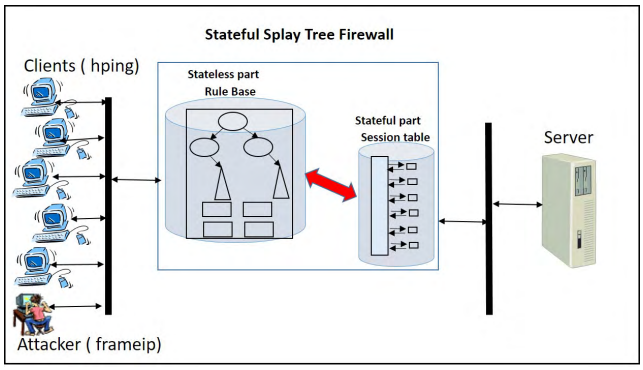

FIGURE 13. Experiment test bed.

VI. PERFORMANCE EVALUATION

The test environment of the splay tree firewall and the proposed session table architecture is deployed with a modified version of Linux CentOS 7.3. Modifications were made to Linux kernel using C language. The testbed consists of six clients and a server, as shown in Fig. 13. All machines use Intel Core i7 with 2.6 GHz CPU and 16 GB RAM. Clients and server are connected to the firewall via Gigabit LAN links. Within the server, two virtual machines are created to serve as Web servers. Clients generate a number of simultaneous queries to request data from Web servers. Packet generator hping3 [44] is used in five of the clients to generate traffic and corresponding processing time is computed. In order to measure the processing time, packet arrival time is recorded in the beginning of the hook_fun(); then again end time is recorded before accepting or dropping the packets, return NF_ACCEPT and return NF_DROP. Attacker machine uses packet generator frameip [45] to generate DoS attack traffic. In all experiments case studies, rules that allow http traffic for the first server is placed at the beginning of the rule base. Whereas, rules allowing traffic for the second server are used in DoS experiments. All experiments are conducted for 10 independent trials, then the averages are plotted.

In the first study case, two different experiments are performed to illustrate the impact of the proposed session table architecture on the splay tree firewall packet processing time. In the first experiment, the session table is removed from the test bed. That is, stateless splay tree firewall is activated. In this case, all incoming traffic is treated as isolated traffic, and filtered without any background history of related packets. Whereas, in the second experiment, proposed session table module is enabled to activate the dynamic statefulness behavior of the splay tree firewall using the proposed session architecture. In both cases, the clients generate the same number of simultaneous TCP connections per second in a continuous and a constant interval of time starting at 10 and ending at 1,000 connections per second. Fig. 14 shows how much extra time is need by the stateful splay tree firewall to filter packets and perform session operations. Stateful splay tree firewall shows an average packet processing time ranges between 0.7 and 0.9 ms depending on whether connection's

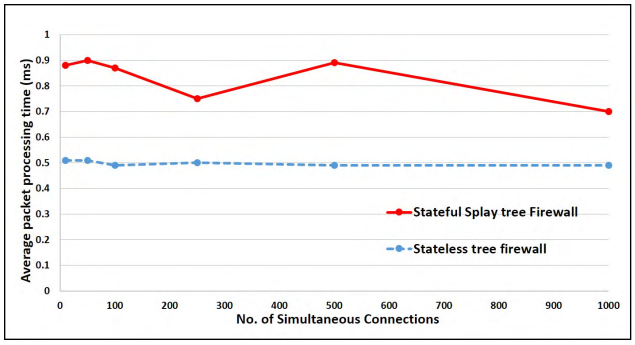

FIGURE 14. Average processing time for stateless and stateful splay tree firewall.

entry has to be inserted on the session table or not, while for stateless splay tree firewall it is almost around 0.5 ms. The extra time added by the stateful firewall is expected; as stateless firewalls are generally faster than stateful firewalls, but less secure. In our case, this extra time is insignificant compared to the security provided using the proposed session table architecture.

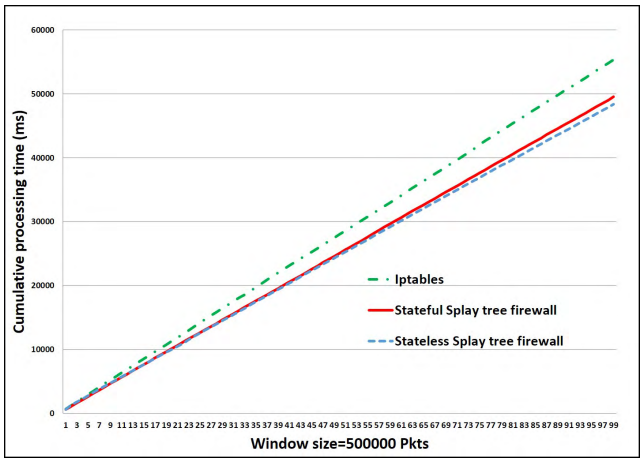

FIGURE 15. Cumulative processing time.

In the second case study, three different experiments are conducted to calculate cumulative packet processing time for IPtables with connection tracking module enabled, splay tree firewall with proposed session table architecture module enabled, and stateless splay tree firewall. A window size of 500 K packets is used, and the cumulative packet processing time is calculated for each system separately over 100 windows. From Fig. 15, Stateful splay tree firewall shows a gain about 18% compared to IPtables. This time gain is due to the use of one hash slot per connection, less hash functions usage, and fast timeout processing in the proposed session table architecture compared to the structures used in IPtables connection tracking system. On the other hand, the stateless splay tree firewall shows a gain of 2.2% compared to stateful splay tree firewall. This gain is negligible, which proves the superior structure of the proposed session table architecture that does not add significant overhead to the system due to connections tracking and related calculation operations.

In the third case study, four experiments are conducted to compare stateful splay tree firewall with the proposed session table module enabled and IPtables. These experiment studies both systems' speed against normal packet traces shown in Fig. 4, 5 and 11. To summarize, there are four different packet traces for both systems:

Trace #1 (First packet is accepted by both firewalls): First packet → Verify against splay tree/ IPtables listed rules → Accept → Store relative packet information in session table → Allow

Trace #2 (First packet is denied by both firewalls): First packet → Verify against splay tree/ IPtables listed rules → Deny → Drop

Trace #3 (Non-first packet is accepted by both firewalls): Non-First packet → Lookup process against Proposed session table/ connection tracking system → Exist → Allow

Trace #4 (Non-first packet is denied by both firewalls): Non-First packet → Lookup process against Proposed session table/ Connection tracking system → Not exist → Drop

Hping command with –S parameter is used to generate first packets requesting connections that suits traces 1 and 2 as follows:

 # hping3 -a 10.10.10.3 -p 80 -S -s+1 -d 64 -i u1000.

This command will generate SYN packets with increased base source port by one using packet size of 64 bytes. This packet size was chosen in order to generate the maximum traffic rate. The parameter −i specifies the time interval for each packet. In this case, 100 packets for second. For traces 3 and 4 the same command is used, but SYN flag is not set.

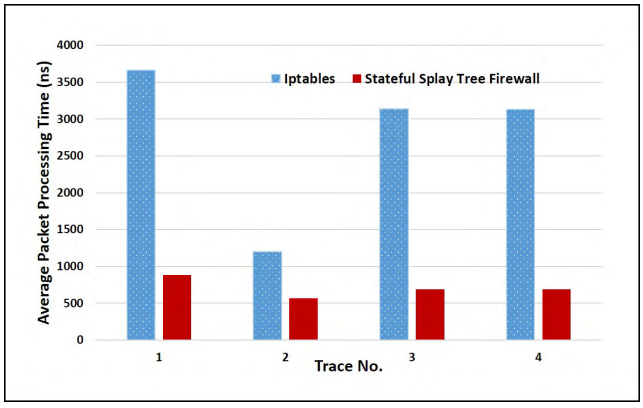

FIGURE 16. Average processing time for each trace for stateful splay tree firewall and IPtables.

Packets that suit each one of the four traces are sent for both systems, and corresponding packet processing times for 10 trials are recorded in nanoseconds. Recorded times are written to a file using PRINTK, and average packet processing time for each trace is calculated separately for each system. Results are displayed in Fig. 16. Stateful splay tree firewall shows lower average packet processing time than IPtables for all the four traces. Trace 1 has the most higher

average packet processing time for both systems. This is due to the time required for verifying packet header against filtering rules then storing packet information in hash tables for both systems. However, Stateful splay tree firewall completed trace 1 by 76% faster than IPtables. The most relevent factors that degrade the performance of IPtables in this trace is the sequential search for its listed rule; specially for increased number of rules. In addition, hashing calculation is performed two times for both packet directions, and packet information is stored twice in the two created nodes. On the other hand, stateful splay tree firewall uses the proposed session table architecture along with filtering rules structured using splay tree with O(log n) amortized time, where n is the number of splay tree nodes [5] . Theoretically, Traces 3 and 4 are the same for each system. These traces focus on comparing the performance of the proposed session table architecture and the connection tracking system apart from the packet filtering method used. In contrast, trace 2 focuses on comparing the filtering method used on both systems apart from the session architecture used. Hence, trace 2 takes less time as no hashing calculation is required. To conclude, stateful splay tree firewall shows a gain of 75% as an average for all the four traces over IPtables.

In the fourth case study, different DoS attacks experiments are conducted to study the behavior of stateful splay tree firewall and IPtables. Open source frameip [45] is used to craft DoS attacks traffic. Generally, attacks may target a specific firewall rule or its session table. The worst form of rule attack is the complexity-algorithmic attack that specifically target bottom rules and degrade considerably the firewall performance. Firewall session table attacks target to fill the session table with illegitimate entries preventing legitimate flows to be established.

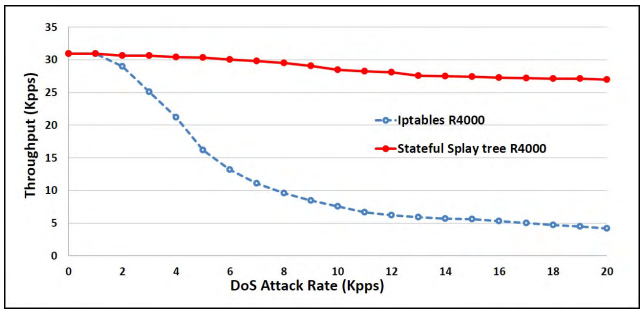

FIGURE 17. Statful splay tree firewall and IPtables throughput for different DoS attack rates targeting rule 4000.

To study the effect of the DoS rule attack, a ruleset is created using 5K rules. The rules that allow normal clients request for server 1 are positioned at the beginning of the rule set, whereas the rule for attacker request for server 2 is placed at position 4000. In between, there are dummy UDP rules with random Src-IP. Then, stateful splay tree firewall and IPtables are subjected to DoS attacks of different rates targeting rule 4000. Fig. 17 shows the throughput of both systems when the DoS attack targets filtering rule at position 4000. IPtables shows a dramatic speed decrease of 86%, when DoS attack rate reaches 20 Kpps. This reflects time spent in sequential search; specially for large number of rules. Whereas, for the stateful splay tree firewall, this attack is prevented without session table intervention using early rejection and acceptance of packets, and splay filters reordering processes, as explained in section 5.6.A. This will keep the firewall almost at steady state with slightly speed degradation of 12.7% at 20 Kpps attack rate.

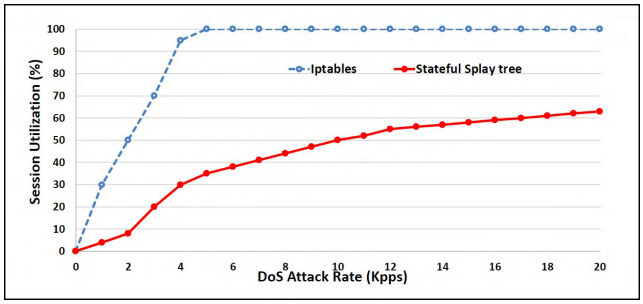

FIGURE 18. Statful splay tree firewall and IPtables session utilization for different SYN attack rates.

Likewise, to study the effect of DoS attacks targeting firewall session table, two experiments are performed on stateful splay tree firewall and IPtables to determine session allocation against different attack rates. SYN attack traffic targeting server 2 is crafted using frameip. Rule that allows traffic to server 2 in this experiment is positioned in the beginning of the rule base, immediately after rules allowing normal traffic to server 1. The reason is that, we want to study the robustness of the proposed session table against regressive demand of session allocation request, and the speed in which the session table tends to reach its maximum utilization, without intervention of the rule position influence. As shown from Fig. 18, IPtables reaches 100% session utilization at 5 kpps. At this particular point, it is observed from Fig. 17 that IPtables throughput is dropped off by almost 50%. While for stateful splay tree firewall, session utilization kept beyond 50% with slight increase until reaching 17 kpps. At 20 Kpps, session utilization reaches ∼63% with an average increase of 42%. This is due to attack mitigation using the separation of Session_ST and Session_ID in the proposed architecture, as explained in section 5.6.B. It is important to mention here that the stateful splay tree firewall results for this experiment differ depending on time in which the results are collected. Since fake session removal using Session_ST depends on the kernel timer object. As time increase, timeout attribute in Session_ST will decrease, and the proposed algorithm will free more session entries. Thus, session utilization is reported after 1, 2, 3 and 4 seconds of the attack. The average result for each attack rate is plotted in the stateful splay tree firewall graph, shown in Fig. 18.

Finally, ACK flood attack is launched using frameip to study the effect of mitigating non- first packet attacks by verifying firstly the splay tree filters, explained in section 5.6.C.

Once the ACK stability test indicates that the stateful splay tree firewall is under non-first packet attack, packets arrive at the firewall may follow one of the following traces shown in Fig. 12:

Trace #5 (Non-first packet is denied by splay Tree): Non-First packet → verify against splay tree → Deny → Drop. This trace is similar to trace 2, except that trace 2 is for SYN packet, while trace 5 is for ACK packet.

Trace #6 (Non-first packet is accepted by splay tree and entry exists in the session table): Non-First packet → verify against splay tree → Accept→ verify against session table → Exist → Allow

Trace #7 (Non-first packet is accepted by splay tree and entry does not exist in the session table): Non-First packet → verify against splay tree → Accept→ verify against session table → Not Exist → Drop.

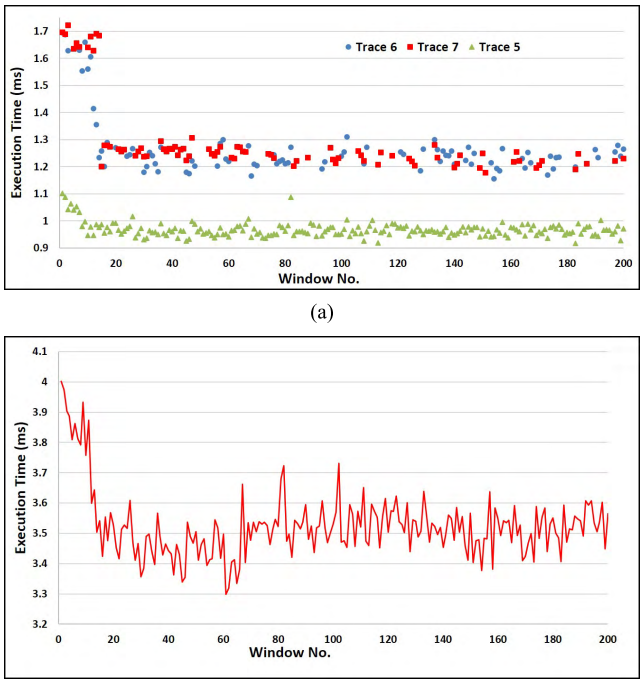

FIGURE 19. Stateful splay tree firewall under non-first packet attack when ACK stability test is enabled (a) and disabled (b).

Fig. 19 (a) and 19 (b) show packet processing time for 500 packets sent with one second interval for 200 windows, when stateful splay tree firewall ACK stability test is enabled and disabled, respectively. In case of disabling the stability test, all ACK flood attack packets will be verified against session table using expensive hashing and session table operations, which require more processing time by the firewall. While, when enabling stability test, ACK flood attack packets will be distributed among traces 5, 6 and 7. Where, packets are verified by the splay tree filters and most were got rejected, saving hashing calculation. The result shows that average packet processing time for traces 6 and 7 are 1.27 and 1.31 ms, which are 24.1% and 25.5% larger than trace 5. However, when stability test is disabled, average packet processing time increased to 3.5 ms, which is 2.6 times more than when stability test is enabled. This means that the firewall needs more than twice the processing power to handle the attack. This is based on the fact that if a non-first packet is denied by splay tree filters no need to further check the session table, because no chance that this packet information to have an entry in the hash table.

VII. CONCLUSION

In this paper, we have proposed a session table architecture and then integrated it with a splay tree firewall to provide a complete statefulness solution. The proposed session table architecture uses one hash slot per connection to save memory space, reduce hash computation, and increase firewall concurrent connections. The session hash table is expanded vertically be reducing the number of buckets using the same memory space to resolve collisions. Moreover, session entry is separated into two different data structures to improve session lookup and enhance timeout processing.

Experiments conducted in this study have shown that the proposed session architecture enhances the packet filtering process compared to IPtables, without adding significant overhead. Furthermore, we studied the impact of different DoS attacks and showed that some of these attacks (such as splay filter high matching and high non-matching attacks) can be mitigated using the nature of the splay tree without session table intervention. While others (such as first packet attacks) can be mitigated using the proposed session architecture, where Session_ST is used to clear massive barged connections trying to fulfill the session table. However, for non-first packets attacks (such as ACK flood attack), an ACK stability test is proposed where the system first checks the splay filters, reducing hashing calculation time by 66%. In future work, we plan to extend the proposed session table architecture to include network address translation (NAT) and quality of service (QoS). Usually, the stateful packet filtering of routers carries out first NAT translation to map between internal and external addresses. Then, session table lookup process is performed. Afterward, QoS classification is done and priority value will be set. Finally, the routing table is looked up and the packet is forwarded accordingly to the next hop. All these operations perform redundant packet header manipulations during lookup processes, which obviously wastes computing resources and increases processing delay. Thus, to increase processing capacity, the needed information for NAT and QoS classification can be integrated with Session_ID attributes. Hence, all needed information will be available with a single search process [48]–[50].

REFERENCES

[1] S. Northcutt, L. Zeltser, S. Winters, K. Kent, R. W. Ritchey, Eds., *Inside Network Perimeter Security: Stateful Firewalls*, 2nd ed. Indianapolis, IN, USA: Sams, Mar. 2005.

[2] Avishai Wool. *Packet Filtering and Stateful Firewalls*. Accessed: Apr. 28, 2018. [Online]. Available: http://www.eng.tau.ac.il/~yash/hinsec-171.pdf

[3] Hank and Foo. (2016). *Stateful Firewalls*. [Online]. Available: http://docplayer.net/3420645-Stateful-firewalls-hank-and-foo.html

[4] Z. Trabelsi and S. Zeidan, "Multilevel early packet filtering technique based on traffic statistics and splay trees for firewall performance improvement," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Ottawa, ON, Canada, Jun. 2012, pp. 1089–1093.

[5] Z. Trabelsi, S. Zeidan, M. M. Masud, and K. Ghoudi, "Statistical dynamic splay tree filters towards multilevel firewall packet filtering enhancement," *Comput. Secur.*, vol. 53, pp. 109–131, Sep. 2015.

[6] J. G. Alfaro, N. Boulahia-Cuppens, and F. Cuppens, "Complete analysis of configuration rules to guarantee reliable network security policies," *Int. J. Inf. Secur.*, vol. 7, no. 2, pp. 103–122, 2008.

[7] A. Hari, S. Suri, and G. Parulkar, "Detecting and resolving packet filter conflicts," in *Proc. INFOCOM*, Tel Aviv, Israel, Mar. 2000, pp. 1203–1212.

[8] E. S. Al-Shaer and H. H. Hamed, "Discovery of policy anomalies in distributed firewalls," in *Proc. INFOCOM*, Hong Kong, Mar. 2004, pp. 2605–2616.

[9] L. Yuan, H. Chen, J. Mai, C.-N. Chuah, Z. Su, and P. Mohapatra, "FIREMAN: A toolkit for firewall modeling and analysis," in *Proc. IEEE Symp. Secur. Privacy*, Berkeley, CA, USA, May 2006, pp. 199–213.

[10] M. G. Gouda and A. X. Liu, "A model of stateful firewalls and its properties," in *Proc. 35th Int. Conf. Dependable Syst. Netw. (DSN)*, Jun./Jul. 2005, pp. 128–137.

[11] L. Buttyán, G. Pék, and T. V. Thong, "Consistency verification of stateful firewalls is not harder than the stateless case," *Inf. Commun. J.*, vol. LXIV, pp. 1–7, Nov. 2009.

[12] W. M. Fitzgerald, S. N. Foley, and M. Ó. Foghlú, "Network access control interoperation using semantic Web technique," in *Proc. WOSIS*, 2008, pp. 26–37.

[13] F. Cuppens, N. Cuppens-Boulahia, J. Garcia-Alfaro, T. Moataz, and X. Rimasson, "Handling stateful firewall anomalies," in *Proc. 27th IFIP Int. Inf. Secur. Privacy Conf. (SEC)*, Heraklion, Greece, Jun. 2012, pp. 174–186.

[14] J. Garcia-Alfaro, F. Cuppens, N. Cuppens-Boulahia, S. Martinez, and J. Cabot, "Management of stateful firewall misconfiguration," *Comput. Secur.*, vol. 39, pp. 64–85, Nov. 2013.

[15] C. Basile and A. Lioy, "Analysis of application-layer filtering policies with application to HTTP," *IEEE/ACM Trans. Netw.*, vol. 23, no. 1, pp. 28–41, Feb. 2015

[16] J. Fong, X. Wang, Y. Qi, J. Li, and W. Jiang, "ParaSplit: A scalable architecture on FPGA for terabit packet classification," in *Proc. IEEE 20th Annu. Symp. High-Perform. Interconnects (HOTI)*, Aug. 2012, pp. 1–8.

[17] O. Erdem and A. Carus, "Multi-pipelined and memory-efficient packet classification engines on FPGAs," *Comput. Commun.*, vol. 67, pp. 75–91, Aug. 2015.

[18] S. Hager, F. Winkler, B. Scheuermann, and K. Reinhardt, "MPFC: Massively parallel firewall circuits," in *Proc. 39th Annu. IEEE Conf. Local Comput. Netw.*, Sep. 2014, pp. 305–313.

[19] H. Hamed, A. El-Atawy, and E. Al-Shaer, "On dynamic optimization of packet matching in high-speed firewalls," *IEEE J. Sel. Areas Commun.*, vol. 24, no. 10, pp. 1817–1830, Oct. 2006.

[20] H. Hamed, A. El-Atawy, and E. Al-Shaer, "Adaptive statistical optimization techniques for firewall packet filtering," in *Proc. IEEE INFOCOM*, Apr. 2006, pp. 1–12.

[21] A. El-Atawy, T. Samak, E. Al-Shaer, and H. Li, "Using online traffic statistical matching for optimizing packet filtering performance," in *Proc. IEEE INFOCOM*, May 2007, pp. 866–874.

[22] A. El-Atawy, E. Al-Shaer, T. Tran, and R. Boutaba, "Adaptive early packet filtering for defending firewalls against DoS attacks," in *Proc. IEEE INFOCOM*, Apr. 2009, pp. 1–9.

[23] W. Wang, R. Ji, W. Chen, B. Chen, and Z. Li, "Firewall rules sorting based on Markov model," in *Proc. Int. Symp. Data Privacy E-Commerce*, Nov. 2007, pp. 203–208.

[24] W. Wang, H. Chen, J. Chen, and B. Liu, "Firewall rule ordering based on statistical model," in *Proc. Int. Conf. Comput. Eng. Technol.*, Jan. 2009, pp. 185–188.

[25] Z. Trabelsi, L. Zhang, and S. Zeidan, "Dynamic rule and rule-field optimisation for improving firewall performance and security," *IET Inf. Secur. J.*, vol. 8, no. 4, pp. 250–257, Jul. 2013.

[26] Z. Trabelsi, L. Zhang, S. Zeidan, and K. Ghoudi, "Dynamic traffic awareness statistical model for firewall performance enhancement," *Comput. Secur.*, vol. 39, pp. 160–172, Nov. 2013.

[27] X. Li, Z.-Z. Ji, and M.-Z. Hu, "Stateful Inspection firewall session table processing," in *Proc. Int. Conf. Inf. Technol., Coding Comput. (ITCC)*, vol. 2, Apr. 2005, pp. 615–620.

[28] X. Li, Z. Ji, and M. Hu, "Session table architecture for defending SYN flood attack," in *Proc. 7th Int. Conf. Inf. Commun. Secur. (ICICS)*, 2005, pp. 220–230.

[29] T. Chomsiri, X. He, P. Nanda, and Z. Tan, "A stateful mechanism for the tree-rule firewall," in *Proc. IEEE 13th Int. Conf. Trust, Secur. Privacy Comput. Commun. (TrustCom)*, Sep. 2014, pp. 122–129.

[30] T. Chomsiri, X. He, P. Nanda, and Z. Tan, "Hybrid tree-rule firewall for high speed data transmission," *IEEE Trans. Cloud Comput.*, Apr. 2016

[31] S. Yoon, B. Kim, J. Oh, and J. Jang, "H/W based stateful packet inspection using a novel session architecture," *Int. J. Comput.*, vol. 2, no. 3, pp. 310–319, 2008.

[32] B. Xu, F. He, Y. Xue, and J. Li, "Architecture-Aware session lookup design for inline deep inspection on network processors," *Tsinghua Sci. Technol.*, vol. 14, no. 1, pp. 19–28, Feb. 2009.

[33] P. N. Ayuso, "Netfilter's connection tracking system," *USENIX Mag.*, vol. 31, no. 3, pp. 40–45, 2006.

[34] K. Salah, K. Elbadawi, and R. Boutaba, "Performance modeling and analysis of network firewalls," *IEEE Trans. Netw. Service Manag.*, vol. 9, no. 1, pp. 12–21, Mar. 2012.

[35] H. Kim, J.-H. Kim, I. Kang, and S. Bahk, "Preventing session table explosion in packet inspection computers," *IEEE Trans. Comput.*, vol. 54, no. 2, pp. 238–240, Feb. 2005.

[36] (Sep. 2015). *Denial-of-Service Attacks Feature Guide for Security Devices*. [Online]. Available: http://www.juniper.net/techpubs/en_US/junos12.../security-attack-denial-of-service.pdf

[37] *Maximizing Firewall Availability: Techniques on Improving Resilience to Session Table DoS Attacks, Version 1.8*, Jul. 2002.

[38] Juniper.net. (2008). *Protecting the Network From Denial of Service Floods*. [Online]. Available: https://jncie.files.wordpress.com/2008/09/801003_protecting-the-network-from-denial-of-service-floods.pdf

[39] Redwolfsecurity.com. (Dec. 2016). *The Next BIG Thing: 'Small' DDoS Attacks are Often Hardest to Block*. [Online]. Available: https://www.redwolfsecurity.com/small-ddos-attacks/

[40] Math.tutorvista.com. *Chi Square Test*. Accessed: Apr. 28, 2018. [Online]. Available: http://math.tutorvista.com/statistics/chi-square-test.html

[41] A. Anand and B. Patel, "An overview on intrusion detection system and types of attacks it can detect considering different protocols," in *Proc. Int. J. Adv. Res. Comput. Sci. Softw. Eng.*, vol. 2, no. 8, pp. 94–98, Aug. 2012.

[42] Corero.com. *Common DDoS Attack Types*. Accessed: Apr. 28, 2018. [Online]. Available: https://www.corero.com/resources/glossary.html

[43] S. Nemade, M. Kumar, and Z. Jamaluddin, "A novel method for early detection of SYN flooding based DoS attack in mobile ad hoc network," *Int. J. Eng. Trends Technol.*, vol. 7, no. 4, pp. 187–191, Jan. 2014.

[44] Hping.org. (2006). *Hping Packet Generator*. [Online]. Available: http://www.hping.org

[45] Frameip.com. (2018). *Frameip Packet Generator*. [Online]. Available: http://www.frameip.com

[46] A. Czubak and M. Szymanek, "Algorithmic complexity vulnerability analysis of a stateful firewall," in *Proc. 37th Int. Conf. Inf. Syst. Archit. Technol. (ISAT-Part II)*, in Advances in Intelligent Systems and Computing, vol. 522. Springer, Sep. 2017, pp. 77–97, doi: 10.1007/978-3-319-46586-9_7.

[47] A. X. Liu, A. R. Khakpour, J. W. Hulst, Z. Ge, D. Pei, and J. Wang, "Firewall fingerprinting and denial of firewalling attacks," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 7, pp. 1699–1712, Jul. 2017.

[48] P. Rengaraju, S. S. Kumar, and C.-H. Lung, "Investigation of security and QoS on SDN firewall using MAC filtering," in *Proc. Int. Conf. Comput. Commun. Inform. (ICCCI)*, Jan. 2017, pp. 1–5.

[49] M. Mostafa, A. A. El Kalam, D. Minuta, and C. Fraboul, "QoS-aware firewall session table," in *Proc. Int. Conf. Risks Secur. Internet Syst.*, Sep. 2011, pp. 1–7.

[50] J. Pettit and T. Graf. (2014). *Stateful Connection Tracking & Stateful Nat*. [Online]. Available: http://www.openvswitch.org/support/ovscon2014/17/1030-conntrack_nat.pdf

[51] B. Jenkins. *A Hash Function for Hash Table Lookup*. Accessed: Mar. 17, 2016. [Online]. Available: http://www.burtleburtle.net/bob/hash/doobs.html

ZOUHEIR TRABELSI received the Ph.D. degree in computer science from the Tokyo University of Technology and Agriculture, Japan. He is currently a Professor of information security with the College of Information Technology, UAE University. His research interests include network security, intrusion detection and prevention, firewalls, TCP/IP covert channels, and information security education.

SAFAA ZEIDAN received the B.Sc. degree (Hons.) in computer engineering from the University of Sharjah, United Arab Emirates. She is currently a Research Assistant with the College of Information Technology, United Arab Emirates University. She has around 13 publications in well-known conferences and journals. Her research interests focus mainly on firewall optimization techniques during normal and attack situations.

KHALED SHUAIB received the B.E. and M.S. degrees in electrical engineering from The City College of New York in 1991 and 1993, respectively, and the Ph.D. degree in electrical engineering/communication networks from the Graduate Center, The City University of New York, in 1999. Since 2002, he has been with the College of Information Technology (CIT), United Arab Emirates University, where he is currently an Associate Professor. During his career at CIT, he held various administrative positions, including a Network Engineering/Networking Track Coordinator from 2004 to 2009, an Associate Dean from 2009 to 2010, an Assistant Dean for Students from 2011 to 2013, and currently he is the Coordinator of three tracks (Information Security, Enterprise Systems, and e-Commerce). He has also been serving as the Director of the Cisco Academy, United Arab Emirates University since 2004. He has over 65 refereed publications in journals and conferences and two U.S. patents. His research interests are in the areas of communication networks, network security, and smart grid.

KHALED SALAH received the B.S. degree in computer engineering with a minor in computer science from Iowa State University, USA, in 1990, and the M.S. degree in computer systems engineering and the Ph.D. degree in computer science from the Illinois Institute of Technology, USA, in 1994 and 2000, respectively. He has over 10 years of industrial experience in embedded systems and software and firmware development. He was with the Department of Information and Computer Science, King Fahd University of Petroleum and Minerals (KFUPM), Saudi Arabia, for 10 years. He joined Khalifa University in 2010, and is teaching graduate and undergraduate courses in the areas of cloud computing, computer and network security, computer networks, operating systems, and performance modeling and analysis. He is currently an Associate Professor with the Department of Electrical and Computer Engineering, Khalifa University, United Arab Emirates. He was a recipient of the Khalifa University Outstanding Research Award in 2014/2015, the KFUPM University Excellence in Research Award in 2008/2009, and the KFUPM Best Research Project Award in 2009/2010, and also a recipient of the departmental awards for distinguished research and teaching in prior years.