ECTI Transactions on Computer and Information Technology
Journal homepage: https://ph01.tci-thaijo.org/index.php/ecticit/
Published by the ECTI Association, Thailand, ISSN: 2286-9131

High-speed Firewall Rule Verification Technique Improves Throughput Performance for IP Version 6

Suchart Khummanee[1], Umaporn Saisangchan[2], Kritsanapong Somsuk[3] and Sarutte Atsawaraungsuk[4]

ABSTRACT

Throughput performance of firewalls depend on the execution speed of verify rules. Internet Protocol Version 6 (IPv6) and IPv4 ruleset memory requirements differ and affect rule access and execution time in a wide range of common firewalls. This paper contributes a high-speed firewall named FW6 to execute rules for IPv6 with constant O(1) access time. FW6 consumes optimal $O(n_{bit})$ memory for 64-bit architectures. Results are based on actual performance evaluations in conjunction with other high-speed firewalls (IPSets, IPack, and F^3), comparing metrics such as processing time, memory consumption and throughput. Throughput measurements in IPv6 TCP/UDP packet trials (across ruleset and window sizes) show FW6 significantly outperforms IPSets. The trials have shown that FW6 improves throughput performance over IPSets by 0.44% (mean) and 0.39% (median) across all test variables. Nevertheless, the results suggest similarity and a minor performance increase by FW6 over IPSets. In addition, FW6 and IPSets throughputs are similar to IPack and F^3 in IPv4 ruleset execution comparisons. As a result, FW6 can be used to replace previous high-speed firewalls.

Article information:

Keywords: Firewall, Firewall Rule Verification, Throughput Performance, IP version 6 (IPv6), Optimality Analysis

Article history:
Received: June 8, 2022
Revised: August 3, 2022
Accepted: August 15, 2022
Published: August 27, 2022 (Online)

DOI: 10.37936/ecti-cit.2022163.248690

1. INTRODUCTION

Firewalls [1] control unauthorized access to resources over a computer network, such as shared data, servers (WWW, Database, DHCP, and DNS), network devices (Routers and Switches), etc. They control access to resources from users by enforcing. Basically, a rule format consists of two parts: a condition term and an action [2, 3]. The condition term consists of five fields: source address (Source IP: SIP), source port (Source Port: SP), destination address (Destination IP: DIP), destination port (Destination Port: DP), and protocol (PRO). The action (ACT) has only one stated either *accept or deny*. The firewall rule format is presented in Table 1.

Table 1: *Example of IPv4 firewall rules.*

Rule No.	Condition					Action
	SIP	SP	DIP	DP	PRO	ACT
1	192.168.1.0/24	*	*	80	TCP	*accept*
2	192.168.1.0/29	*	*	22	TCP	*accept*
3	192.168.1.0/24	*	200.0.0.5	*	*	*deny*
4	*	*	*	345	UDP	*deny*
5	*	*	*	*	*	*deny*

Rule no.1 allows a group of source IP addresses (SIP = 192.168.1.0/24 = 256 IPs) to connect to any destination website provider (DIP = *, DP = 80) using the TCP protocol (PRO = TCP). Rule no.3 does not allow a group of 256 source IP addresses to communicate with a destination IP address (200.0.0.5). The last rule is always set to drop unspecified communications.

The speed of verifying rules determines firewall performance. According to previous research [1], there are three levels of speed to execute rules: rules with sequential data structures [4, 20], rules with tree structures [5, 6], and rules with hashed data structures [1, 7, 8]. The speed of processing rules with a sequential structure is Big-O(n), where n is the num-

[1,2]The authors are with the Department of Computer Science, Faculty of Informatics, Mahasarakham University, Maha sarakham, Thailand, E-mail: suchart.k@msu.ac.th and umaporn@msu.ac.th
[3]The author is with the Department of Computer and Communication Engineering, Faculty of Technology, Udon Thani Rajabhat University, UDRU, Udon Thani, Thailand, E-mail: kritsanapong@udru.ac.th
[4]The author is with the Department of Computer Education, Udon Thani Rajabhat University, Udon Thani, Thailand, E-mail: sarutte@udru.ac.th

ber of rules. The speed of processing tree-style rules is Big-O($\log_n$). Last, rules with hashed data structures have the fastest processing time of Big-O(1). Even if firewalls deployed using hashed data structures have the highest rule-verifying speed, they still consume much memory. For example, IPsets [7, 8] consumes memory equal to Big-O(2^n), where n is the memory required by the administrator. IPack [7] spends the memory space as Big-O(n) to handle rules, where n is the number of rules. Last, FFF [7] allocates Big-O(n) memory spaces to store rules, where n is the number of bits (one binary bit per one rule). However, IPack and FFF are suitable for implementation with IP version 4 (IPv4), with a maximum data bit of 104 bits, as illustrated in Fig 1(a). For IP version 6 (IPv6), they are not yet available as the number of bits of IPv6 is up to 296 bits ($\approx$ 284% of IPv4), as shown in Fig 1(b). The number of bits directly affects the processing time and the memory space to store rules.

In this paper, we have developed a novel technique to verify rules for IP version 6. This technique has the same speed of Big-O(1), but consumes only Big-O(n_{bit}) of memory, where n is the number of rules and uses an optimized block size for IPv6 rule execution on 64bit architectures.

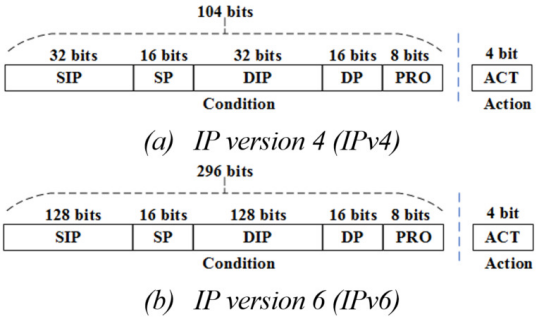

Fig.1: *Number of rule bits of IPv4 and IPv6.*

The paper is structured as follows. The section 2 discusses the background and related work including firewall operations and rules, rule anomalies, high-speed firewall processing models, and IPv4 and IPv6 models. Section 3 presents our contribution. The design and development of the high-speed firewall is Section 4. Section 5 evaluates our proposed firewall against other high-speed firewalls. Lastly, Section 6 is a summary of this paper.

2. BACKGROUND AND RELATED WORK

2.1 Fundamentals of Firewalls

A ***firewall*** is a popular security tool installed between the Internet and private networks, as shown in Fig 2.

Its main function is to filter information (commonly known as packets) that flows in and out of the network. Packets are filtered based on rules defined by the network policy. For example, a company has a

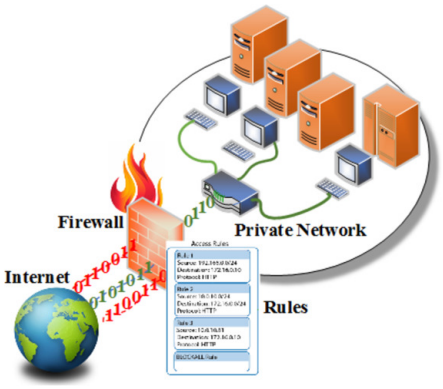

Fig.2: *Firewall installation and functionality*

policy that all employees cannot use social media such as Facebook during working hours, except for public relations officers (PR). This policy may have more than one rule when converted to the firewall rules. In this case, assume that the public relations officer's IP addresses (SIP) are 192.168.1.10-15, and the Facebook IP address (DIP) is 157.240.10.35. Therefore, this policy can be converted to two rules as follows:

Rule no. 1: allow PR officers to use Facebook, and

Rule no. 2: deny employees from using Facebook.

These rules convert to actual firewall rules as in Table 2.

Table 2: *Example of establishing rules.*

Rule No.	Condition					Action
	SIP	SP	DIP	DP	PRO	ACT
1	192.168.1.10-15	*	157.240.10.35	80	TCP	*accept*
2	*	*	157.240.10.35	80	TCP	*deny*

Where Facebook uses the TCP protocol and port number 80, both rules cannot be reversed in order as they can lead to conflicts [2, 3] and loss of meaning [9]. In fact, any two rules conflict only if their conditional terms overlap and have different actions. For example, in Table 2, rule no. 1 and rule no. 2 conflict (rule no. 1 $\subset$ rule no. 2), but the firewall can still process the rules. On the other hand, if rule no. 2 is swapped over rule no. 1, rule no. 1 will never be executed (rule no. 1 is absolutely overshadowed by rule no. 2). There are six types of firewall rule conflicts: Shadowing, Correlation, Generalization, Redundancy, Irrelevancy [3], and Semantics loss [9]. Methods for detecting and resolving rule conflicts are proposed in [2-5, 9, 10].

2.2 High-speed Firewall Models

High-speed firewalls in this paper refer to firewalls that have a rule execution speed of Big-O(1). The scope of evaluation is limited to firewall applications available for the Linux-based open source operating systems. In this paper, three types of high-speed firewalls are discussed: IPSets, IPack, and FFF (F^3) as follows.

IPSets [8]: It is an open-source utility that works with IPTables (Core firewall) [11] on Linux operating system. IPSets improved the speed of rule verification of IPTables from O(n) to O(1), where n is the number of rules. It speeds up the execution of rules by applying hashing techniques. However, firewall administrators need to be proficient in rule grouping for smaller rule sets, and it takes much memory to store rules (O(2^n)).

IPack [1]: It was developed to resolve all of IPSets problems and also consumes less memory (O(n)). Moreover, it can also detect and eliminate rule conflicts. However, the IPack has a problem with the time for establishing the data structure (O(n^2)) because it is necessary to compact the data contained in a three-dimensional structure into a one-dimensional structure.

F^3 [7]: It improved the memory consumption of IPack from O(n) to O(n_{bit}) using the bit mapping technique, where nbit is the size of one rule per bit. In addition, F^3 also reduced the time to establish rules from O(n^2) to O(n). It introduces a new technique known as FIRST MATCH - FIRST EXECUTE (FMFE) to eliminate rule conflicts. This technique prunes all obscured rules. Although F^3 has the best performance, it is only designed for IPv4, and it is not yet compatible with IPv6.

In summary, IPSets is currently compatible with IPv6, but it shares the other usability limitations as IPv4. As for the IPack, it does not support IPv6 like F^3 yet.

2.3 Basic understanding of IPv4 and IPv6

An IP address is used to identify a computer device on a computer network. It is a unique address. IP addresses being used today are version 4 (IPv4 [12]), and they are 32 bits in size. Therefore, the total number of available IPv4 addresses is 2^{32} (4,294,467,295 IPs). IPv4 is divided into four sets of eight bits, each separated by a dot (.), 192.168.1.10. Therefore, the available IP numbers range between 0.0.0.0 and 255.255.255.255. Unfortunately, the number of IPv4 addresses is insufficient, and it will be retired very soon.

IPv6 is replacing IPv4, and it will completely replace IPv4 soon. IPv6 [12, 13] has several features that are better than IPv4:

- IPv6 increases the size of address numbers from 32 bits to 128 bits, as illustrated in Fig 1; thus, the maximum number of available IP addresses is 3.4 * 10^{38},
- IPv6 header is a less complex and fixed size, making processing faster than IPv4,
- IPv6 enhances QoS capabilities for real-time service,
- IPv6 Increases the size of the MTU from 567 bytes to 1,280 bytes, resulting in more efficient throughput,
- IPv6 does not need Network Address Translation (NAT) like IPv4; consequently, it can be traced backward.

There are four types of IPv6 addresses: Unicast, Anycast, Multicast, and Loopback address, as shown in Fig 3. Unicast is used for communicating data from one device to another in a network (also known as one-to-one communication). Anycast allows multiple devices on the network to share the same IP address. The data is sent to the nearest device based on the user's request location (one-to-closest of many). Multicast enables a device to send the data to a specified set of devices in different subnetworks, known as one-to-many communication. Last, the Loopback address is an internal address that routes back to the local system (also called localhost).

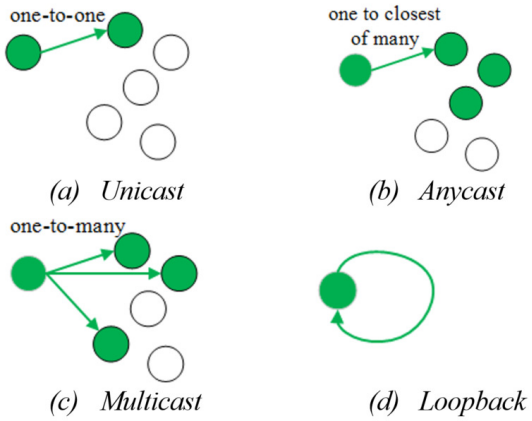

Fig.3: *Type of IPv6 Address.*

The size of IPv6 addresses is 128 bits, divided into eight groups of hexadecimal numbers, each with 16 bits and separated by colons (":") as follows.

0000:0000:0000:0000:0000:ffff:7f00:0001 (IPv6) or 127.0.0.1 of IPv4.

IPv6 addresses are written in the form IPv6-address/prefix-length. For example, 2001:000:00:0000 :cd30:0000:0000:0000:0001/64, where /64 is the prefix-length. The prefix length [14, 15] is a decimal number that indicates the number of bits used to calculate its network address. For example, 2001:0000:0000:cd30:0000:0000:0000:0001/64 has a 64-bit network address of 2001:0000:0000:cd30::, and its 64-bit host address is 0000:0000:0000:0001. IPv6 addresses are difficult to write and remember. Therefore, they can be written in abbreviated form, e.g. "2001:0:0:cd30::1/64". The rules for abbreviated writing IPv6 are as follows.

1. If any group in IPv6 is all zeros, it can be replaced by a single zero (0). For example, "2001:**0000**:**0000**:cd30:**0000**:**0000**:**0000**:0001/64" can be rewritten as "2001:**0**:**0**:cd30:**0**:**0**:**0**:1/64".

2. If there are consecutive groups of zeros, they can be shortened with double colons (::), and it can only be used once (usually choose the

longest consecutive group of zeros). For example, "2001:**0**:**0**:cd30:**0**:**0**:**0**:1/64" can be written as "2001::cd30:0:0:0:1/64" or "2001:0:0:cd30::1/64". For ease of understanding, Table 3 shows the implementation differences between IPv4 and IPv6 based on the RFC 2373 (original version of IPv6 [14]) and RFC 4291 (current version [15]).

Table 3: *Comparison of usage between IPv4 and IPv6.*

IPv4 address		IPv6 address	
Type	**Subnet mask**	**Type**	**Subnet prefix**
Public IP address	*.*.*.*/n	Global unicast address	::/n (RFC 4291)
Private IP address	10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16	Unique local address (ULA)	FC00::/7
Link local address	169.254.0.0/16	Link local address	FE80::/10
Multicast address	224.0.0.0/4	Multicast address	FF00::/8
Loopback address	127.0.0.1	Loopback address	::1/128

From Table 3, all public IPv4 addresses (*.*.*.*/n) can be used except Private IP, Link-local, Multicast, and Loopback addresses, where /n is the number of bits for subnetting. Similar to IPv4, all IPv6 addresses (Global unicast address or ::/n) are available except Unique local, Link-local, Multicast, and Loopback address.

IPv6 subnet prefix: Generally, Global unicast IPv6 is always designed to be compatible with 64 bits or /64 prefix lengths. Other subnet prefix lengths (not /64) should be used with caution as they affect the capabilities of IPv6's design, such as Neighbor Discovery (ND), Secure Neighbor Discovery (SEND), privacy extensions, parts of Mobile IPv6, Protocol Independent Multicast, and others.

Currently, for regular customers, Internet Service Providers (ISP) always provide /48 or /58 subnet prefixes (large companies may get subnet prefixes from /32 to /40). The case where a customer has been allocated a subnet prefix of size /48 is illustrated in Fig 4. The customer can establish their own networks (Subnet ID) with up to 65,536 networks (2^{16}), with 2^{64} hosts for each network (Interface ID).

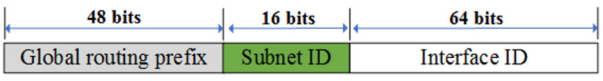

Fig.4: *IPv6 prefix length and subnetting.*

Convert IPv4 to IPv6: Conversion from IPv4 to IPv6 is done by converting each IPv4 octet to hexadecimal. For example, 192.168.1.10 converts to C0.A8.01.0A. Subsequently, the converted hexadecimal numbers are rearranged with each group of 16 bits separated by :. For example, C0.A8.01.0A rearranges to ::C0A8:010A. The IPv4 rules from Table 1 can be converted to IPv6 rules as shown in Table 4.

Table 4: *Example of IPv6 firewall rules.*

Rule No.	Condition					Action
	SIP	**SP**	**DIP**	**DP**	**PRO**	**ACT**
1	fe80::c0a8:100/120	*	::	80	TCP	*accept*
2	fe80::c0a8:100/125	*	::	22	TCP	*accept*
3	fe80::c0a8:100/120	*	2002::c800:5/128	*	*	*deny*
4	::	*	::	345	UDP	*deny*
5	::	*	::	*	*	*deny*

3. RESEARCH CONTRIBUTION

Table 5 shows the overall performance of all high-speed firewalls. We found that F3 had the highest performance. However, it still had two flaws: the time for eliminating the conflicting rules (O(n)), and it does not support IPv6. The contribution of this research:

1. Improve F^3 to support IPv6 rules,
2. Eliminate conflicting rules by verifying rules simultaneously; it eliminates rule conflicts with Big-O(1),
3. Apply an optimum memory representation of IPv6 rule sets for access and execution.

Table 5: *High-speed firewall characteristics.*

Firewall metric	Firewall name		
	IPSets	**IPack**	**F^3**
Time for rule verifying	**O(1)**	**O(1)**	**O(1)**
Time for establishing rule structure	$O(n^2)$	$O(n^2)$	**O(n)**
Time for eliminating conflict rules	**No**	O(n)	O(n)
Consumed memory	$O(2^n)$	O(n)	**$O(n_{bit})$**
IPv4 support (Class A, B, C and D)	C	All	**All**
IPv6 support	**Yes**	No	No
Ability to detect rule conflicts	No	**Yes**	**Yes**
Rule management skills	High	**Low**	**Low**
Technique for matching rules	Hash	Index	Index
Complexity of data structures	Low	High	**Low**

4. HIGH-SPEED FIREWALL DESIGN AND DEVELOPMENT FOR IPV6

There are four steps in the design and development of our high-speed firewall for IPv6:

1. Calculating the optimum data block size to store firewall rules.
2. Mapping IPv6 rules to bit data blocks.
3. Verifying rules against packets (matching algorithm).
4. Implementing our IPv6 firewall for throughput performance evaluation.

Each step is explained in detail in the following sections.

4.1 Optimum data block size for collecting IPv6 firewall rules

F^3's technique [7] is based on mapping rules to blocks of bits, with each block-size equal to 256 (2^8) slots, as shown in Fig 5. Each slot is the same size as n bits, where n is the number of rules. Each block stores each data field of the rules. For example, SIP_1 keeps the 1st octet of source IP addresses, SIP^2 stores the 2^{sd} octet of source IP addresses, etc. The number of data blocks of F^3 is 10 blocks (4 * SIP + 4 * DIP + 2 * DP). The source port field (SP) is not considered because they are always random ports. ACCEPT and DENY block keep the action of the rules.

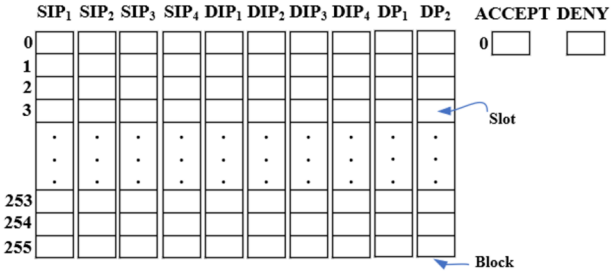

Fig.5: *The bit memory blocks for F^3.*

The memory size consumed by F^3 is 10 * 256 * n_{bit} + 2 * n_{bit}. Assume the number of firewall rules is 100 (n = 100). Thus, F^3's memory consumption is 32.03 Kilobytes (KB). However, these block sizes are not optimal for IPv6 because the number of data bits is four times that of IPv4. Therefore, we need to recalculate the optimal block size for the IPv6 using Equation (1).

$$\frac{NB * NS * n_{bit} + 2 * n_{bit}}{8} \quad (1)$$

NB is the number of data blocks. NS is the number of slots per block, and n is the number of rules. 8 is the number of bits per byte. Calculations for blocks and slots suitable for storing IPv6 rules are shown in Table 6. We split the bit size of the data (128 bits) into four groups (Number of bits per block), each of which is 4, 8, 16, and 32 bits. Consequently, the number of blocks is 32, 16, 8, and 4 bits, respectively. In the case of the number of blocks being 32, the number of slots is 16. Another example when the number of blocks is 16, the number of slots is 256. The results obtained from Table 6 show that the optimal memory size for collecting IPv6 rules requires the maximum number of blocks and the least number of slots. For example, in the first row of Table 6, there are 128 rules to test. The optimal memory is 0.0082 MB, which is achieved by setting the number of blocks to 32 and the number of slots to 16.

Although using many blocks and the small number of slots resulted in optimal memory usage, the speed of execution of rules (processing speed) gradually decreased. The speed of bit processing is directly dependent on the number of blocks. Given the processing speed of one bit per second, the number of rules is 128 (the first row in Table 6), and the number of blocks is 32, 16, 8, and 4, the processing speed (Equation (2)) will be 4096, 2048, 1024, and 512 seconds (sec).

$$NB * PSB * R \quad (2)$$

PSB is the processing speed of bits (in this example, one bit per second). R is the number of firewall rules. For example, if the number of blocks is 32, then the processing time of each bit is 1 sec, and the number of rules is 128. So the processing speed equals 4096 sec (32 * 1 * 128).

Determining the optimal block size to store IPv6 rules requires low memory consumption and low processing time. Based on the memory consumption and processing speed from Table 6, we find that the optimal number of blocks is either 8 or 16. If the number of blocks is 8, a firewall with 0.5 gigabytes (GB) or more memory can process it (for storing 16,384 rules). On the other hand (number of blocks as 16), the firewall machine must have at least 2.5 GB of memory to process 16,384 rules.

In this research, we chose the number of blocks equal to 8 because current firewalls can handle this amount of memory consumption, and it also provides a high speed of execution of rules. Therefore, the total memory size for handling 16,384 IPv6 rules in this research is 2.41 GB as follows:

Total memory = (8 * SIP) + (8 * DIP) + (1 * DP) + (1 * PRO) + (2 * ACT) = (8 * 65536 * 16384) + (8 * 65536 * 16384) + (1 * 65536 * 16384) + (1 * 65536 * 16384) + (2 * + 16384) = 2.41 GB.
Example memory blocks for collecting the generated IPv6 rules are shown in Fig 6.

4.2 Mapping IPv6 rules to bit memory blocks

This section maps the firewall rules from Table 4 into bit memory blocks, as shown in Fig 6.

The first step is to set the default data bits of all slots in each block to zero. The number of bits in each slot equals the number of rules. For example, in Table 4, the number of rules is five; thus, the initial data bit in each slot is 00000.

The second step is to separate the source and destination IPv6 addresses into eight groups (SIP_{1-8} and DIP_{1-8}), each of which is equal to 16 bits. For example, rule number 1 has SIP = fe80::c0a8:100/120 (/120 = 256 addresses). Therefore, the SIP is separated into eight groups: ZIP_1 = fe80, ZIP_2 = 0000, ZIP_3 = 0000, ZIP_4 = 0000, ZIP_5 = 0000, ZIP_6 = 0000, ZIP_7 = c0a8, and ZIP_8 = 0100 – 01ff (256 addresses). Then map rule number 1 (00001) to the address of the slot in each block accordingly. For example, the first group (ZIP_1) of rule number 1 is mapped to slot number $fe80_{16}$ (65,15210). This slot

Table 6: *Evaluating optimal block sizes for storing IPv6 rules.*

Number of rules	Number of bits per block	Number of blocks	Number of slots per block	Cconsumed memory (MB)	Processing speed (1 bit per second)
128 (2^7)	4	32	16	0.0082	4,096
	8	16	256	0.0655	2,048
	16	8	65,536	8.3886	1,024
	32	4	4,294,967,296	274,877.91	512
256 (2^8)	4	32	16	0.0164	8,192
	8	16	256	0.1311	4,096
	16	8	65,536	16.777	2,048
	32	4	4,294,967,296	549,755.81	1,024
512 (2^9)	4	32	16	0.0328	16,384
	8	16	256	0.2622	8,192
	16	8	65,536	33.554	4,096
	32	4	4,294,967,296	1,099,511.63	2,048
1,024 (2^{10})	4	32	16	0.0657	32,768
	8	16	256	0.5245	16,384
	16	8	65,536	67.109	8,192
	32	4	4,294,967,296	2,199,023.26	4,096
2,048 (2^{11})	4	32	16	0.1315	65,536
	8	16	256	1.0490	32,768
	16	8	65,536	134.21	16,384
	32	4	4,294,967,296	4,398,046.51	8,192
4,096 (2^{12})	4	32	16	0.2631	131,072
	8	16	256	2.0981	65,536
	16	8	65,536	268.43	32,768
	32	4	4,294,967,296	8,796,093.02	16,384
8,192 (2^{13})	4	32	16	0.5263	262,144
	8	16	256	4.1963	131,072
	16	8	65,536	536.87	65,536
	32	4	4,294,967,296	17,592,186.05	32,768
16,384 (2^{14})	4	32	16	1.052672	524,288
	8	16	256	8.392704	262,144
	16	8	65,536	1,073.745	131,072
	32	4	4,294,967,296	35,184,372.09	65,536

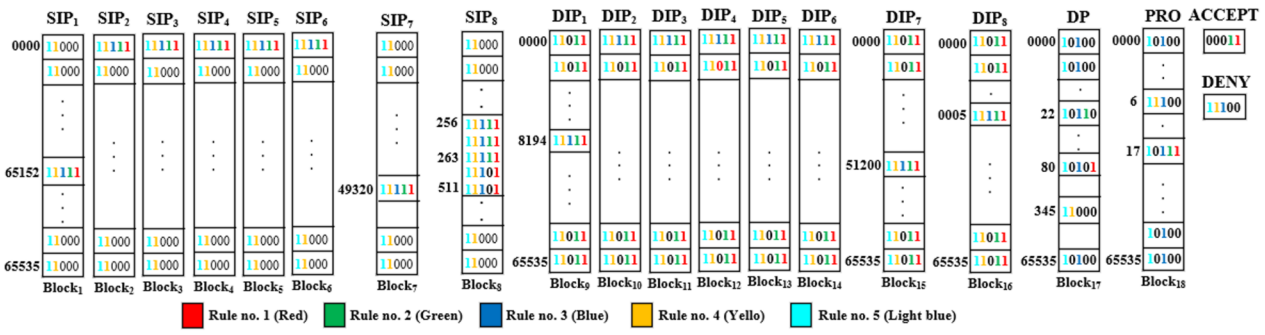

Fig. 6: *Mapping all IPv6 rules to bit data blocks.*

is recorded as 00001 (rule no. 1). The slot address of group number 2 (ZIP_2) is 00000 (010). Thus, the address is recorded as 00001. For group numbers 3-6 (ZIP_3-ZIP_6), mapping is the same as group number 2. An address of slot number c0a8 ($19{,}320_{10}$) of block number 7 (ZIP_7) saves bit data as 00001. The source IP addresses of the last group (ZIP_8) ranges from 0100 to 01ff (256 addresses). Thus, slots numbered from 0100 (256_{10}) to 01ff (511_{10}) of the 8^{th} block (ZIP_8) are all recorded as 00001, as shown in Fig 6.

The next step is mapping the destination IPv6 addresses (DIP_{1-8}). Refer to the DIP of rule no. 1 from Table 4, :: is all destination IPv6 addresses. As a result, every slot in each DIP block is recorded as 00001. The destination port number of rule no. 1 is 80 (webserver); thus, it is mapped to the 80th slot of block number 17. At this address, 00001 is saved. TCP and UDP are defacto protocols for transferring data over the IPv4 network [16]. They are also used on the IPv6 network. The port number of TCP is 17, and UDP is 6 [17]. Based on rule no. 1, slot no. 17 of block no. 18 (PRO) is set to 00001. Finally, the action of rule no. 1 is *accept*. Thus, the ACCEPT block is set to 00001, and DENY block is set to 00000. The

rest of the rules (rules 2-5) are processed the same as rule no. 1. The data bits of each slot are set according to the rule number. For example, rule no. 1 is 00001, rule no. 2 is 00010, rule no. 3 is 00100, respectively, as illustrated in Fig 7. When all the rules from Table 4 are mapped into bit data blocks, the result is shown in Fig 6.

Fig.7: *Setting rules into bit data slots.*

4.3 Matching packets against IPv6 rules

This procedure shows the algorithm used for matching packets to already mapped rules.

Let ACTION be a decision variable, and TEMP be a temporary variable. The size of the data bits of ACTION and TEMP is equal to the number of rules. The ACTION is initially set to 1 for every bit, and TEMP is initially set to 0 for every bit. For example, from Table 4, the number of rules has a total of 5. Thus, ACTION is set to 11111, and TEMP is set to 00000. Suppose a single packet incoming to the firewall consists of SIP = fe80::c0a8:105, DIP = 2002::c800:64, DP = 80, and PRO = TCP (17). How is this packet processed by the firewall? (To pass or drop). Initially, SIP and DIP of the packet are separated into eight groups, each of which is equal to 16 bits. An example of SIP: SIP_1 = fe80, SIP_2 = 0000, SIP_3 = 0000, SIP_4 = 0000, SIP_5 = 0000, SIP_6 = 0000, SIP_7 = c0a8, and SIP_8 = 0105. An example of DIP: DIP_1 = 2002, DIP_2 = 0000, DIP_3 = 0000, DIP_4 = 0000, DIP_5 = 0000, DIP_6 = 0000, DIP_7 = c800, and DIP_8 = 0064. Next, convert the data of SIP_1-SIP_8 and DIP_1-DIP_8 from hexadecimal to decimal to use as references to the data in each slot of each block. The sample data after converting SIP_1 - SIP_8 to decimal: SIP_1 = 65152 ($fe80_{16}$), SIP_2 = 0, SIP_3 = 0, SIP_4 = 0, SIP_5 = 0, SIP_6 = 0, SIP_7 = 49320 ($c0a8_{16}$), and SIP_8 = 261 (0105_{16}). The sample data after converting DIP_1 - DIP_8 to decimal: DIP_1 = 8194 (2002_{16}), DIP_2 = 0, DIP_3 = 0, DIP_4 = 0, DIP_5 = 0, DIP_6 = 0, DIP_7 = 51200 ($c800_{16}$), and DIP_8 = 100 (0064_{16}).

The next step fetches the data stored in each slot of each block with the address calculated from the previous step. Fetched data is then saved in the TEMP. For example, slot no. 65152 of block no. 1 (SIP1) has data of 11111, so TEMP is saved as 11111. Next, compute ACTION and TEMP by using the bitwise AND operator. As a result, ACTION is equal to 11111 (ACTION = ACTION AND TEMP by ACTION = 11111 AND TEMP = 11111). Next, fetches data from slot no. 0 of block no. 2 (SIP_2), which is 11111, and saves it to TEMP. It is then executed with ACTION by bitwise AND operator as in the previous step. The procedure for calculating SIP_1-SIP_8 and DIP_1-DIP_8 is shown in Table 7.

Table 7: *A packet matching against rules.*

Rule	Operation	ACTION variable	TEMP variable
1	Initialize ACTION and TEMP.	11111	00000
2	Fetch data from address 65152 (Slot no. 65152, Block no. 1: SIP1), and save to TEMP	11111	11111
3	ACTION = ACTION **AND** TEMP	11111	11111
4	Fetch data from address 0 (Block no. 2: SIP2), and save to TEMP	11111	11111
5	ACTION = ACTION **AND** TEMP	11111	11111
6	Fetch data from address 0 (Block no. 3: SIP3), and save to TEMP	11111	11111
7	ACTION = ACTION **AND** TEMP	11111	11111
8	Fetch data from address 0 (, Block no. 4: SIP4), and save to TEMP	11111	11111
9	ACTION = ACTION **AND** TEMP	11111	11111
10	Fetch data from address 0 (Block no. 5: SIP5), and save to TEMP	11111	11111
11	ACTION = ACTION **AND** TEMP	11111	11111
12	Fetch data from address 0 (Block no. 6: SIP6), and save to TEMP	11111	11111
13	ACTION = ACTION **AND** TEMP	11111	11111
14	Fetch data from address 49320 (Block no. 7: SIP7), and save to TEMP	11111	11111
15	ACTION = ACTION **AND** TEMP	11111	11111
16	Fetch data from address 261 (Block no. 8: SIP8), and save to TEMP	11111	11111
17	ACTION = ACTION **AND** TEMP	11111	11111
18	Fetch data from address 8194 (Block no. 9: DIP1), and save to TEMP	11111	11111
19	ACTION = ACTION **AND** TEMP	11111	11111
20	Fetch data from address 0 (Block no. 9-14: DIP9-14), and save to TEMP	11111	11111
21	ACTION = ACTION **AND** TEMP	11111	11111
22	Fetch data from address 51200 (Block no. 15: DIP15), and save to TEMP	11111	11111
23	ACTION = ACTION **AND** TEMP	11111	11111
24	Fetch data from address 100 (Block no. 16: DIP16), and save to TEMP	11111	11011
25	ACTION = ACTION **AND** TEMP	11011	11011
26	Fetch data from address 80 (Block no. 17: DP), and save to TEMP	11011	10101
27	ACTION = ACTION **AND** TEMP	10001	10101
28	Fetch data from address 17 (Block no. 18: PRO), and save to TEMP	10001	10111
29	ACTION = ACTION **AND** TEMP	**10001**	10111

Table 7 shows the packet-to-rule matching process. The final result of the calculation is saved in the ACTION (Step 29), which is 10001. The last step is ACTION (10001) **AND** ACCEPT (00011), and ACTION (10001) **AND** DENY (11100). The result has two values, 00001 and 10000. 00001 means packet matched to rule no. 1, which can pass (*accept*). The result of 10000 means the last rule of the firewall is to drop every packet (implicit deny).

Consider a case of conflicting rules, e.g. a packet contains SIP = fe80::c0a8:105, DIP = 2002::c800:5, DP = 80, and PRO = TCP. The final result of the execution saved in ACTION is 10101. When **AND** with ACCEPT (00011) and DENY (11100), the result is 00001 and 10100. It shows that rule no. 1 and no. 3 conflict with each other. The solution to this problem is always to give the utmost importance to the preceding rule, in which case rule no. 1 is executed.

4.4 Implementing our IPv6 firewall

The data structure for storing rules is a bit type. Thus, we declare a bit-field structure in the C/C++ language.

```
STRUCT {
  TYPE [MEMBER_NAME]: WIDTH;
};
```

TYPE is an integer type that determines how a bit-field's value is interpreted. The type may be signed int or unsigned char. MEMBER_NAME is the name of the bit-field. WIDTH is the number of bits in the bit-field. For example, declaring the data structures in C/C++ language for storing SIP, DIP, DP, PRO, ACCEPT, and DENY from Table 4:

```
#define MAX_SLOT 65536
#define roundup(x) ((x)%64==0?(int)(x)/64:(int)(x))/64+1
#define round(x) ((x)<=0?(int)(1):roundup(x))

    typedef struct {
      unsigned int data;
    } bitBlock;

    NUM_BIT = round(n);
    bitBlock* SIP1-8[MAX_SLOT][NUM_BIT];
    bitBlock* DIP1-8[MAX_SLOT][NUM_BIT];
    bitBlock* DP[MAX_SLOT][NUM_BIT];
    bitBlock* PRO[MAX_SLOT][NUM_BIT];
    bitBlock* ACCEPT[1][NUM_BIT];
    bitBlock* DENY [1][NUM_BIT];

    for (int i=0; i < MAX_SLOT; i++){
      for (int j=0; j < NUM_BIT; j++){
        SIP1[i][j] = malloc(sizeof(bitBlock));
      }}
    ......
```

MAX_SLOT is the maximum number of slots in each block. Due to the limitations of modern computer architectures, when C/C++ requests dynamic memory space (malloc), the memory allocated each time is sixty-four bits (for 64-bit architectures) and thirty-two bits (for 32-bit architectures). Thus, there is no need to declare the bit-field (WIDTH) in the typedef struct because the default of WIDTH is always 64 bits per each memory allocation (even if declared as unsigned char (8 bits)). Therefore, this designed IPv6 firewall uses unsigned int (data) instead of unsigned char because it corresponds to the memory allocation of 64-bit architecture. n is the number of all rules. NUM_BIT is the number of bits representing the number of rules (1 rule per bit). NUM_BIT is calculated by dividing the total number of rules by 64. As a result of division, if there is a remainder, the number of bits is not enough. Therefore, it has to add one to NUM_BIT (NUM_BIT = NUM_BIT + 1). For example, if the number of rules equals 50, the NUM_BIT is 1 (50/64 = 0.7, and rounded up from 0.7 to 1). Another example, if the number of rules equals 225, NUM_BIT is 4 (225/64 = 3.51, rounded up from 3.51 to 4). Notice that NUM_BIT equals 1 means that memory is allocated to 64 bits (Allocated memory = NUM_BIT * 64 bits). The example of allocated memory space corresponds to the rules from Table 4, as shown in Fig 8.

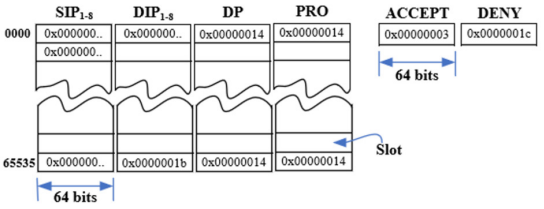

Fig.8: *Memory allocation for IPv6 rules in Table 4 on 64-bit architectures.*

Setting bits in each slot from Fig 6 by C/C++ language can be done as follows:

$$SIP_1[65152][0]\text{->data} = 0x0000001f;$$
$$ACCEPT[0][0]\text{->data} = 0x00000003;$$

In the example above, the bit sequence 0x0000001f (11111) is assigned to slot address number 65152 of SIP_1. Likewise, the ACCEPT block is assigned the value 0x00000003 (00011).

4.5 IPv6 Firewall Development

The core framework of the IPv6 firewall was developed in C/C++ language (GCC 4.4.7) together with GNU Make 3.8 on 64-bit Linux kernel version 2.6, as shown in Fig 9.

The IPv6 firewall framework consists of three main layers: the User space, the Kernel space, and the Physical. The User space layer is a command-line

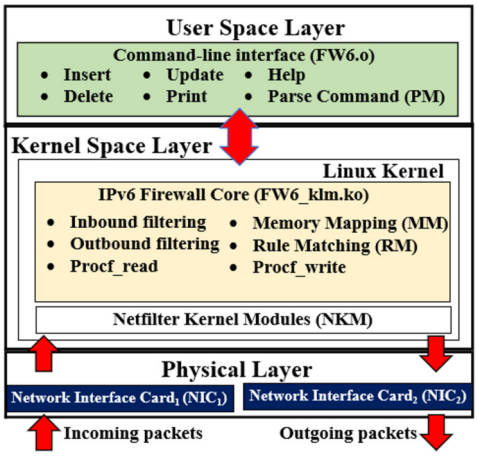

Fig.9: *IPv6 firewall framework (FW6).*

interface. It performs the following functions: taking commands from users, interpreting commands (parse command: PM), displaying results (print), and providing various commands to facilitate an administrator, such as insert, delete, update, and help. The program working in this layer is called FW6.o. The command syntax of the firewall is as follows.

fw6 –in/out interface [–srcip] [–mask] [–dstip] [–mask] [–srcport] [–dstport] [–proto] –action.
Fw6 is the name of the firewall program. –in/out is used to specify the interface for incoming and outgoing packets such as eth0, enpos3, or wlan0. –srcip specifies the source IPv6 address (option = [...]). If the source IP address is not declared, it means every IP address. –mask is the subnet mark such as /128, /120. –dstip determines the destination IP address. –srcport defines the source port number. Usually, the srcport is always random; thus, it is rarely declared. –dstport indicates the destination port number such as 80 (WWW), 53 (DNS), etc. If the dstport is not assigned, its default is all destination ports. –proto is a protocol like TCP or UDP. If the protocol is not defined, its default is all protocols. Last, –action represents the decision of each rule. If any rule decides to allow the packet to pass the firewall, the action of such a rule has a status of accepted (*accept*). Otherwise, the status is blocked (*deny*). For example, creating rules that can execute according to Table 4:

Rule no. 1:
Fw6 –in eth0 –srcip fe80::c0a8:100 –mask /120 –dstip * –dstport 80 –proto TCP –action accept
Rule no. 2:
Fw6 –in eth0 –srcip fe80::c0a8:100 –mask /125 –dstip * –dstport 22 –proto TCP –action accept
Rule no. 3:
Fw6 –in eth0 –srcip fe80::c0a8:100 –mask /120 –dstip 2002::c800:5 –mask /128 –action deny
Rule no. 4:
Fw6 –in eth0 –srcport 345 –proto UDP –action deny
Rule no. 5:
Fw6 eth0 –action deny

The next layer, Kernel space, filters both the incoming and outgoing packets transmitted from the Physical layer (network interface card: NIC) to the Linux kernel using the Netfilter Kernel Module (KVM) [18]. The filtered packets must comply with the defined rules. This layer also maps the defined rules to memory (MM) and matches the packets against the mapped rules (RM). It displays results and handles commands from the administrator (User space) through the procf_read and procf_write file system. The main kernel module acting on this layer is named FW6_klm.ko. Note that the IPv6 must be enabled for use on Linux by configuring "net.ipv6.conf.default.disable_ipv6 = 0" and "net.ipv6.conf.all.disable_ipv6 = 0" commands in the /etc/sysctl.conf file.

The last layer is the Physical layer. This layer is responsible for establishing the firewall and communicating data packets to the networks via the network card (NIC). The incoming packets pass through the first interface (NIC1), and the outgoing packets pass through the second interface (NIC_2). The experimental version of FW6 can be downloaded from this URL: https://github.com/Suchart-k/FW6.

5. TESTBED AND PERFORMANCE EVALUATION

5.1 Configuration criteria and networking settings

In this research, the criteria and network settings for testing firewalls are defined as follows.

The testbed network for testing the firewalls is connected to the Internet network using Ethernet technology, with a bandwidth of 1 Gbps. The tested network is organized into two parts, the server-side and the client-side. The server-side generates the requested packets from the client-side and evaluates the network throughput using IPERF Server software [19]. The client-side (IPERF Client) constantly sends connection requests and receives data from the server. The firewalls to be tested are connect between the server-side and the client-side, as shown in Fig 10. Note that throughput refers to the rate of data transfer from one node to another relative to a unit of time, such as seconds.

Firewall and network configuration criteria for evaluating the performance of firewalls are shown in Tables 8 and 9, respectively.

5.2 Performance evaluation of firewalls

All firewalls are evaluated with four metrics: processing time, memory consumption, throughput, and other characteristics.

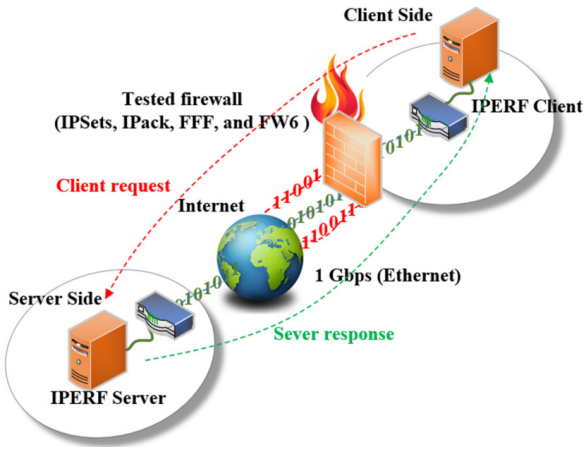

Fig.10: *Network testbed for high-speed firewalls.*

Table 8: *Firewall settings.*

Criteria	Specification
The name of the firewall used in the test	IPSets, IPack, FFF, and FW6
The number of rules to be tested in each epoch	100, 500, 1000, 2000, 3000, 4000, 5000, 10000
The number of rounds used for each epoch	30 rounds per epoch
Protocol used for testing	TCP and UDP
The type of IP address being used for testing	IPv4 and IPv6
Software that supports running firewalls	Linux kernel 2.6 for 64 bits (Ubuntu desktop 20.04), GCC 4.4.7, GNU Make 3.8, and Netfilter
Software used for evaluation (throughput)	IPERF server and IPERF client
Hardware for a running firewall server	Intel Core i-7 2.0 GHz for 64 bits, 4 cores 8 threads, 8 GB RAM
Hardware for a running client	Intel Core i-5 1.6 GHz for 64 bits, 2 cores 4 threads, 4 GB RAM
The number of interface cards	Two cards for incoming interface (NIC_1) and outgoing interface (NIC_2)

Table 9: *Network settings.*

Criteria	Specification
Data communication technology and transmission speed	Ethernet technology, 1 Gbps
Types of data used to test communication	Unicode (Text) and binary packet
Window size (WS) for data communication	16, 32, and 64 Kilobytes (KB)
The MTU size	1,500 bytes
The network interval time	1 sec
The concurrent session	250 connections
The connections per second (CPS)	470 connections
UDP bandwidth transfer	1 Gbps
Router and switching forwarding capacity	100/1,000 Mbps

The processing time: there are two types of processing time:

1. The time to construct the rule structure and,
2. The time to verify the rules.

Refer to Table 5, the time for constructing the rules of IPSets, IPack, F^3: $O(n^2)$, $O(n^2)$, and O(n), respectively. However, since FW6 (our proposed firewall) applies the same method of constructing rules as F^3, FW6 takes time equal to that of F^3, that is O(n). The IPSets rule verifying technique is hashing, so the speed is O(1). Indexing is a method of referencing directly to data in the main memory, resulting in the speed of verifying rules of IPSets, IPack, F^3, and FW6 equal to the hashing methods, that is O(1).

The memory consumption is the main memory used to store firewall rules. The memory is used until the firewall device is reset. The consumed memory shown in Table 5 is for IPv4. However, the space complexity for IPv6 is equivalent to IPv4. That memory consumption of IPSets, IPack, F^3, and FW6 is equal to $O(2^n)$, O(n), $O(n_{bit})$, and $O(n_{bit})$, respectively. However, although the space complexity of IPv4 and IPv6 firewalls are the same, in practice they have some differences, as demonstrated by Equations 3.

$$IPSets = \frac{Keys * Bits * 2(Tables)}{8 * 10^6} \quad (3)$$

The Keys is the number of hashed keys. The Bits is the number of bits used to store the hashed keys. In this case, it is 64 bits. The Tables are Immediate Table (G) and Value Table (V) [7], which are used to store keys after hashing. Keys are produced by Cartesian product (X) operation between rule fields, for example SIP X DP X PRO = fe80::c0a8:100/120 X 80 X 17 = 256 keys. In Table 4, the sum of all keys from rule no. 1 to no. 4 (IPSets does not support the last rule (no. 5) because it is *) equals 7.2E+81 keys. The number of computed keys is more than the IPSets can support. Therefore, the IPSets solution is to group rules with many keys into smaller groups to reduce the number of keys. For example, IPSets does not take fields * and :: into the calculation; thus, the key from Table 4 is reduced to 34,665,244 keys. Consequently, the memory consumed by the IPSets is equal to 554.64 MB. However, the BigO of IPSets is 2^n; thus, it needs to allocate an actual memory size equal 8,192 MB (2^{13}).

The IPack firewall consumes a fixed memory size because the structure used for storing rules is a one-dimensional array [1]. Equation 4 is used for calculating the amount of memory of IPack for IPv6.

$$IPack = \frac{65,536 * Bits + 8,541 * Bits * n}{8 * 10^6} \quad (4)$$

n is the number of rules. Bits is the memory used to store rules. In this example, the memory size for each slot is 64 bits. Therefore, the consumed memory for the rules in Table 4 for IPack is 0.86 MB.

F^3 and FW6 rely on direct access to main memory like IPack. However, since F^3 and FW6 process data in bits, they consume less memory than IPack. Equation 5 calculates the memory of F^3 and FW6.

$$F^3, FW6 = \frac{18 * 65,536 * n + 2 * n}{8 * 10^6} = \frac{1,179,650 * n}{8 * 10^6} \quad (5)$$

n is the number of bits (one bit per rule). For example, if the rule has a number of 5, the memory allocated to F^3 and FW6 is 0.73 MB. In fact, the allocated memory of a computer system in practice is always the same size as the actual memory. For example, in C/C++ for a 64-bit computer, the character variable (CHAR) is allocated 64-bit memory (users know 8 bits). Therefore, F^3 and FW6 allocate real memory equal to (1,179,650*64)/8*106 = 9.43 MB.

An overview of the memory consumption of all tested firewalls is shown in Fig 11. Let the average number of hashed keys per rule for IPSets be 16,384. The test uses only the firewall rule's SIP, DIP, and DP field because IPSets has to reduce the number of keys. The number of bits used to store data is 128 bits. The results showed that the highest memory consumption was for IPSets, the second highest was for IPack, and the lowest was for F^3 and FW6.

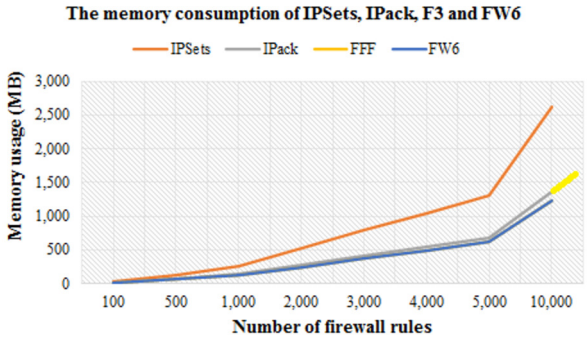

Fig.11: *Compare the memory consumption of high-speed firewalls.*

Throughput is measured by the amount of traffic flowing in and out of the firewall over a unit of time (second) using metrics from Tables 8 and 9. We classify throughput into two types, TCP and UDP. The data used for evaluation was in text and binary packets. For TCP packets, the window size (WS) for testing is set to 16 KB, 32 KB, and 64 KB, respectively. The bandwidth is set to a fixed value of 100 Mbps with the UDP packets. Estimating the throughput of all firewalls was similar since they had a rule verifying speed of O(1). The results are shown in Fig 12 and 19, respectively.

However, IPack and FFF are slightly more performant in their speed to verify rules than other firewalls since they have direct access to in-memory data. In addition, they only work with IPv4, where they have fewer bits of data to be checked than IPSets and FW6. In comparison, the speed of checking rules IPSets and FW6 is used a static test result.

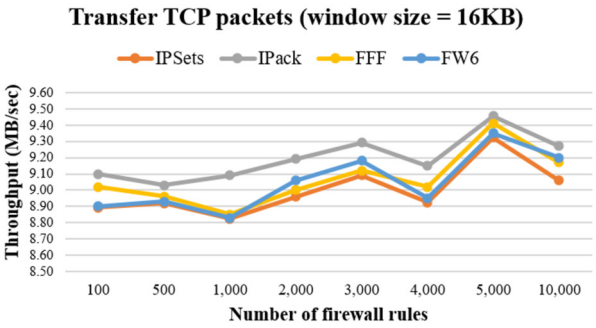

Fig.12: *TCP throughput (text) with WS=16K.*

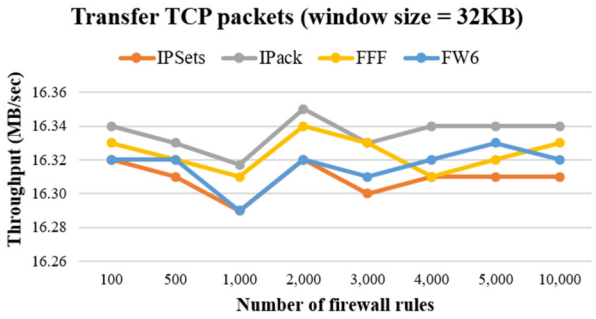

Fig.13: *TCP throughput (text) with WS=32K.*

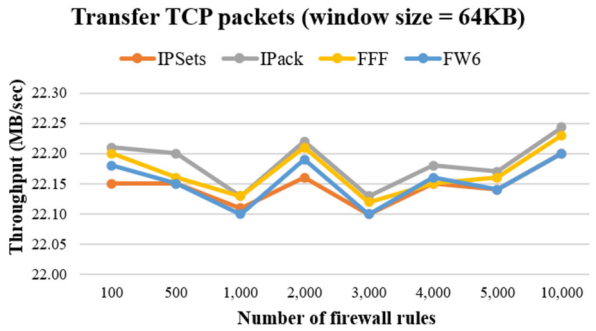

Fig.14: *TCP throughput (text) with WS=64K.*

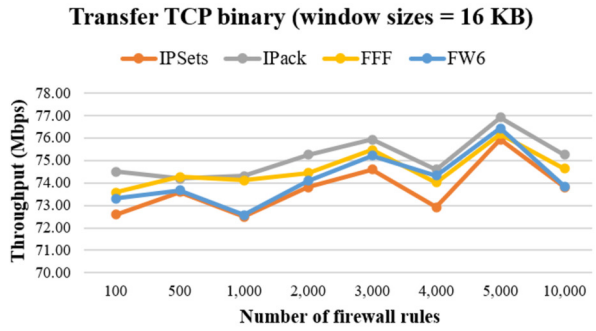

Fig.15: *TCP throughput (binary) with WS=16K.*

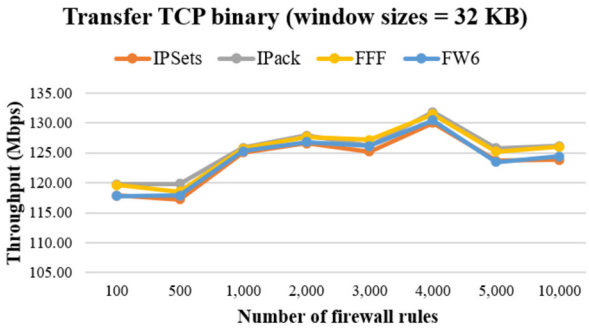

Fig.16: *TCP throughput (binary) with WS=32K.*

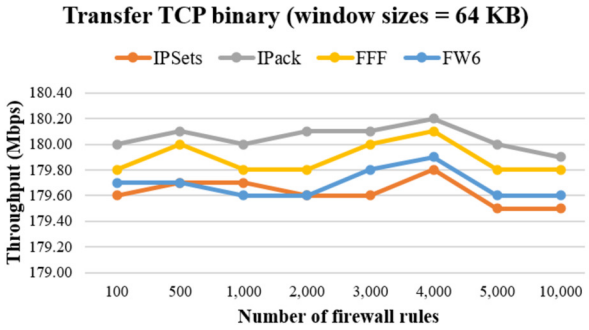

Fig.17: *TCP throughput (binary) with WS=64K.*

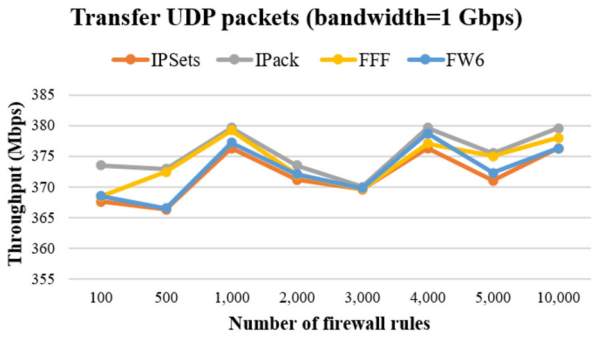

Fig.18: *UDP throughput (text).*

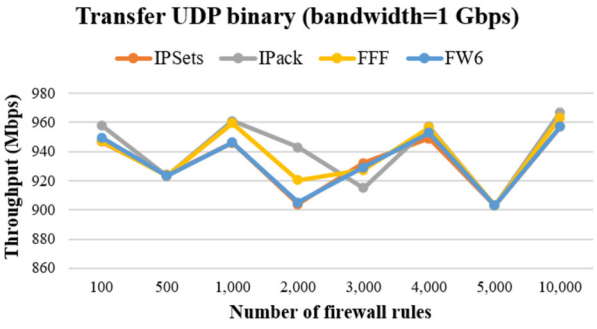

Fig.19: *UDP throughput (binary).*

6. CONCLUSION

This research improves the F3 firewall's ability to process rules for IPv6 with BigO(1) access time, and the result is called FW6. The efficiency of FW6 is compared to IPSets, IPack, and F3 in three categories: processing time, memory consumption, and throughput. The results showed that FW6 consumed the least memory on both IPv4 and IPv6 rules, with a rule verifying speed of BigO(1). Although the throughput of FW6 is slightly less than that of IPack and F3, in terms of IPv4 packets, all firewalls are considered to have the same throughput and are interchangeable. Rule processing efficiency between IPSets and FW6 for IPv6 gives the same result of O(1), however the memory consumption of FW6 is much less than that of IPSets. An overview of firewall performance is shown in Table 10.

Table 10: *Firewall performance comparison.*

	Firewall name			
Firewall metric	**IPSets**	**IPack**	**F^3**	**FW6**
Time for rule verifying	O(1)	O(1)	O(1)	O(1)
Time for building rule structure	$O(n^2)$	$O(n^2)$	O(n)	O(n)
Time to eliminate conflict rule	No	O(n)	O(n)	No
Consumed memory	$O(2^n)$	O(n)	$O(n_{bit})$	$O(n_{bit})$
IPv4 support (Class A-D)	C	Yes	Yes	Yes
IPv6 support	Yes	No	No	Yes
Ability to detect rule conflicts	No	Yes	Yes	Yes
Rule management skills	High	Low	Low	Low
The complexity of data structures	Low	High	Low	Low

Note: IPSets does not recommend making a subnet larger than Class C. In practice, memory allocation of F^3 and FW6 always depends on the memory architecture. For example, declaring a byte variable in C, the operating system always allocates memory equal to 32 bits. (32-bit architecture).

In summary, the analysis of trials has shown that FW6 increases throughput performance by 0.44% (mean) and 0.39% (median) compared to IPSets on IPv6 transmissions. In the IPv4 ruleset execution comparisons, FW6 and IPSets have throughput similar to IPack and F3.

References

[1] S. Khummanee, "IP Packing Technique for High-speed Firewall Rule Verification," *Journal of Internet Technology*, vol. 20, no. 6, pp. 1737-1751, Nov. 2019.

[2] S. Khummanee, P. Chomphuwiset, P. Pruksasri, "Decision Making System for Improving Firewall Rule Anomaly Based on Evidence and Behavior," *Advances in Science, Technology and Engineering Systems Journal*, vol. 5, no. 4, pp. 505-515, Aug. 2020.

[3] S. Khummanee, P. Chomphuwiset, and . P. Pruksasri, "DSSF: Decision Support System to Detect and Solve Firewall Rule Anomalies based on a Probability Approach," *ECTI-CIT Transactions*, vol. 16, no. 1, pp. 56–73, Mar. 2022.

[4] Diekmann, C., Hupel, L., Michaelis, J. et al., "Verified iptables Firewall Analysis and Verification," *Journal of Automated Reasoning*, vol. 61, no. 1, pp. 191–242, 2018.

[5] S. Khummanee, A. Khumseela and S. Puangpronpitag, "Towards a new design of firewall: Anomaly elimination and fast verifying of firewall rules," in *Proc. JCSSE*, pp. 93-98, 2013.

[6] H. Hamed, A. El-Atawy and E. Al-Shaer, "On Dynamic Optimization of Packet Matching in High-Speed Firewalls," in *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 10, pp. 1817-1830, Oct. 2006.

[7] S. Khummanee, P. Songram, and P. Pruksasri, "FFF: Fast Firewall Framework to Enhance Rule Verifying over High-speed Networks," *ECTI-CIT Transactions*, vol. 16, no. 1, pp. 35–47, Mar. 2022.

[8] J. Kadlecsik, "IP set features," Accessed on: Apr.,2021. [Online]. Available: `https://ipset.netfilter.org`

[9] S. Khummanee, "The Semantics Loss Tracker of Firewall Rules," in *Proc. IC2IT*, 2018, pp. 220-231.

[10] C. Togay, A. Kasif, C. Catal and B. Tekinerdogan, "A Firewall Policy Anomaly Detection Framework for Reliable Network Security," in *IEEE Transactions on Reliability*, vol. 71, no. 1, pp. 339-347, March 2022.

[11] S. Suehring, "Linux Firewalls: Enhancing Security with nftables and Beyond," Accessed on: Jan. 29, 2015. [Online]. Available: `https://www.amazon.com/dp/0134000021?tag=uuid10-20.`

[12] J. Pyles, J. L. Carrell, E. Tittel, "Guide to TCP/IP: IPv6 and IPv4," Accessed on: Jun. 14, 2016. [Online]. Available: `https://www.amazon.com/Guide-TCP-IP-IPv6-IPv4/dp/1305946952.`

[13] J. J. Amoss and D. Minoli, "Handbook of IPv4 to IPv6 Transition: Methodologies for Institutional and Corporate Networks," Accessed on: Sep. 19, 2019. [Online]. Available: `https://www.amazon.com/Handbook-IPv4-IPv6-Transition-Methodologies/dp/0367388057.`

[14] R. Hinden, "IP Version 6 Addressing Architecture," *Cisco Systems.*, Accessed on: Jul.,1998. [Online]. Available: `https://www.rfc-editor.org/info/rfc2373.`

[15] R. Hinden, "IP Version 6 Addressing Architecture," *IETF.*, Accessed on: Feb.,2006. [Online]. Available: `https://www.rfc-editor.org/info/rfc4291.`

[16] D. E. Comer, *Internetworking with TCP/IP*, London, WC2R 0RL: Pearson,, pp. 185–246, 2013.

[17] B. A. Forouzan, *TCP/IP Protocol Suite*, United States, New York: McGraw Hill, pp. 375–501, 2009.

[18] J. Kadlecsik and G. Psztor, "Netfilter performance testing" Accessed on: Dec.,2020.[Online]. Available: `https://people.netfilter.org/kadlec/nftest.pdf.`

[19] B. A. Jon Dugan and E. Seth, "IPERF - the ultimate speed test tool for TCP, UDP and SCTP, IPERF testing report 2017" Accessed on: June.,2020. [Online]. Available: `https://iperf.fr/.`

[20] OpenBSD Journal, "PF: Firewall Ruleset Optimization" Accessed on: Sep.,2006.[Online] Available: `https://undeadly.org/cgi?action=front.`

Suchart Khummanee received the B.Eng. degree in Computer Engineering from the King Mongkut's Institute of Technology Ladkrabang, the M.Sc. degree in Computer Science from the Khon Kaen University, and the Ph.D. degree in Computer Engineering from the Khon Kaen University, Thailand. He is currently a full lecturer of Computer Science at the Mahasarakham University, Thailand. His research interests in the network security, computer networks, agricultural robotics, and Internet of things (IoT).

Umaporn Saisangchan received a M.Sc. degree in Computer Science from the Khon Kaen University. She is currently a full lecturer at the Faculty of Informatics, Mahasarakham University (MSU), Thailand. Her research focuses on data mining, algorithms, classification and applications.

Kritsanapong Somsuk is an associate professor at the Department of Computer and Communication Engineering, Faculty of Technology, Udon Thani Rajabhat University, Udon Thani, Thailand. He obtained his M.Eng. (Computer Engineering) from Department of Computer Engineering, Faculty of Engineering, Khon Kaen University, M.Sc. (Computer Science) from Department of Computer Science, Faculty of Science, Khon Kaen University and his Ph.D. (Computer Engineering) from Department of Computer Engineering, Faculty of Engineering, Khon Kaen University. The area of research interests includes computer security, cryptography and integer factorization algorithms.

Sarutte Atsawaraungsuk received the B.Sc. and M.Sc degree in Computer Science from the Khon Kaen University, and the Ph.D. degree in Computer Engineering from the Khon Kaen University, Thailand. Currently, he is an assistant professor at the Department of Computer Education, Faculty of Education, Udon Thani Rajabhat University, Udon Thani, Thailand. His research interests in the Machine learning, image processing and its application.