A Connection Management Protocol for Stateful Inspection Firewalls in Multi-Homed Networks

Jin-Ho Kim, Heejo Lee, and Saewoong Bahk

Abstract: **To provide network services consistently under various network failures, enterprise networks increasingly utilize path diversity through multi-homing. As a result, multi-homed non-transit autonomous systems become to surpass single-homed networks in number. In this paper, we address an inevitable problem that occurs when networks with multiple entry points deploy firewalls in their borders.**

The majority of today's firewalls use stateful inspection that exploits connection state for fine-grained control. However, stateful inspection has a topological restriction such that outgoing and incoming traffic of a connection should pass through a single firewall to execute desired packet filtering operation. Multi-homed networking environments suffer from this restriction and BGP policies provide only coarse control over communication paths. Due to these features and the characteristics of datagram routing, there exists a real possibility of asymmetric routing. This mismatch between the exit and entry firewalls for a connection causes connection establishment failures.

In this paper, we formulate this phenomenon into a state-sharing problem among multiple firewalls under asymmetric routing condition. To solve this problem, we propose a stateful inspection protocol that requires very low processing and messaging overhead. Our protocol consists of the following two phases: 1) Generation of a TCP SYN cookie marked with the firewall identification number upon a SYN packet arrival, and 2) state sharing triggered by a SYN/ACK packet arrival in the absence of the trail of its initial SYN packet. We demonstrate that our protocol is scalable, robust, and simple enough to be deployed for high speed networks. It also transparently works under any client-server configurations. Last but not least, we present experimental results through a prototype implementation.

Index Terms: **Connection management protocol, multi-homed networks, network security, routing asymmetry, stateful inspection firewalls, SYN cookies.**

I. INTRODUCTION

A. Background

The Internet engineering task force (IETF) is extending the AS number size from 2 octets to 4 octets with the network size growing [26]. Enterprise networks increasingly leverage path diversity through multi-homing because a single Internet service provider (ISP) is not enough to provide consistent performance. Multi-homing, which refers to a single network having more than one connection to the Internet, has been used widely for enhancing the reliability of the network connectivity and increasingly used for better networking performance [1], [8]. Today, multi-homed non-transit autonomous systems (ASes) surpass single-homed networks in number [12], [27]. Thanks to its connectivity failure reduction and load balancing features affordable at a modest cost, multi-homing is on such a steep rise in deployment.

An important phenomenon on the Internet is asymmetric routing, which is a real possibility for multi-homed networks. Even before the current advent of multi-homed networks, it was shown that routing paths are often asymmetric in the Internet [18]. Recent studies have shown the prevalence of routing asymmetry in the Internet [9], [10], where the measurements were conducted either at the AS level or at the router level.

Meanwhile, most enterprise networks need to deploy firewalls in their border to protect themselves from illegitimate traffic. Recently, what is called the stateful inspection firewall has become the de facto industry standard. It enables flexible and fine-grained control over incoming traffic, and the filtering decision can be dynamically made based on the need of the outgoing traffic. Namely, a stateful inspection firewall intercepts a packet and updates its internal state table with the extracted connection state information (source and destination addresses and port numbers), based on which the filtering decision is made for the incoming traffic [7]. However, this means that the stateful inspection has a topological restriction: Outgoing and incoming traffic of a connection should pass through a single firewall.

The failure of meeting this restriction due to the possibility of asymmetric routing leads to the connection establishment problem. Multi-homed networks may have multiple entry/exit points (henceforth, MEPs) because of topological restrictions, physical location diversity for reliability and so forth. If an AS is with MEPs, outgoing and incoming flows may go through different firewalls. In this case conventional stateful inspection designed for a single entry point (henceforth, SEP) does not work as desired. One may think that moving SEP firewalls to end hosts or a subnet with a single entry point can solve the problem. However, the placement of firewalls enforcing route symmetry cannot be a fundamental solution for stateful inspection in MEP networks and even it may cause to lose their protection coverage and induce the limited network utility.

In this paper, we address this inevitable problem that occurs when MEP networks deploy firewalls in their border. In particular, we explore a state sharing method between firewalls used

Manuscript received April 10, 2007; approved for publication by Dongman Lee, Division III Editor, July 29, 2008.

This work was supported in part by Seoul R&BD Program, the ITRC program of the Korea Ministry of Knowledge Economy, and Defense Acquisition Program Administration and Agency for Defense Development under the contract 2008-SW-51-IJ-03. This is an extended version of the conference paper published in the proceedings of IEEE ICC, 2004 [13].

J. Kim is with Google, Seoul, Korea.

H. Lee is with the Division of Computer and Communication Engineering, Korea University, Seoul 136-713, Korea, email: heejo@korea.ac.kr.

S. Bahk is with the Department of EE, INMC, Seoul National University, Seoul 151-742, Korea, email: sbahk@snu.ac.kr.

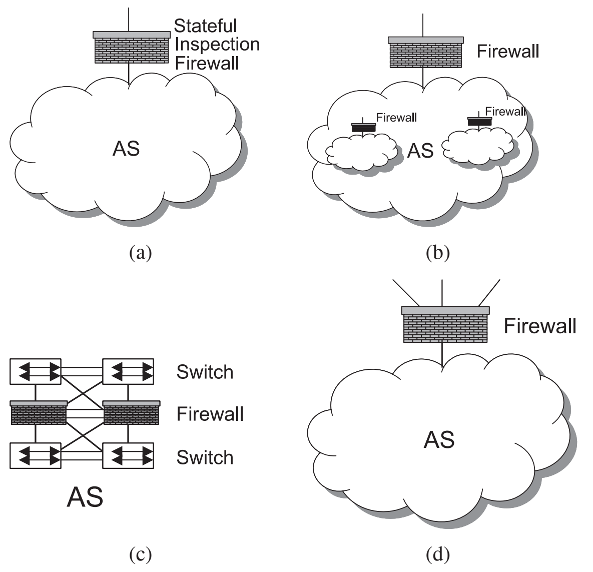


Fig. 1. Single entry point firewalls; (a) stateful inspection firewall, (b) distributed firewalls, (c) high availability setup, and (d) multi-link firewall.

in the MEP networks and the stateful inspection mechanism to filter out packets from suspicious connections.

B. Related Works

We can consider a few approaches to configure a network with firewalls according to the positioning of the firewalls. They include distributed firewalls for enhancing firewall performance, firewalls with high availability setup, and multi-link firewalls for increasing connectivity, which are shown in Fig. 1. The three approaches can be described as follows:

- Distributed firewalls: Firewalls are distributed within an AS. To do stateful inspection with distributed firewalls, all routing paths still need to be symmetric. As conventional firewalls cannot filter any traffic they have not seen, all hosts inside the protected sub network should be trusted. Distributed firewalls were proposed for finer-grained access control against internal threats [2]. In this configuration, security policy is centrally defined but its enforcement is left to an individual endpoint [11].
- High availability setup: With high availability setup, stateful inspection firewalls share connection information with each other through state mirroring. The firewalls can achieve linearly increased throughput by putting them in active mode, and if any one fails, its connections will be taken over by some other one. Duplicated firewalls normally use a dedicated communication line for mirroring state information. This system is effective in SEP environments where a load balancer or a switch distributes traffic over several firewalls. If this system is with MEP environments, a race condition occurs between a SYN/ACK packet arrival and its state mirroring. State mirroring requires processing time and mirroring intervals, and it is hard to use dedicated communication links in MEP environments. Particularly, if a TCP server is located in the same AS with firewalls and a client is outside the network, its SYN/ACK packet will probably arrive at a firewall before state mirroring is completed. Therefore state mirroring loses the race and the establishment of the TCP connection will be dropped or severely delayed [28].
- Multi-link firewalls: Multi-link technology enables an AS with a firewall to be connected with multiple ISPs [14]. Although it can make use of multiple ISPs, all links are merged into one firewall.

Thus, this is not relevant to our considered MEP network. Above approaches are proper for SEP networks, but inadequate for MEP networks. As well, existing security mechanisms have been devised on the basis of the assumption that Internet routes are symmetric at the AS level [5], or at the router level [7]. These approaches are not adequate for the current Internet environments with large degree of routing asymmetry [9], [10].

C. Contributions of This Work

The main contributions of this work are three-fold. First, we introduce the problem of firewalling for an MEP network. Second, we propose a method of performing stateful inspection in the MEP network, and design an effective mechanism for exchanging connection states among the firewalls. The proposed mechanism is scalable such that its complexity is not related to the number of firewalls. Third, through a prototype implementation, we verify that high-speed packet filtering is achievable in the MEP network through minimizing the overhead caused by the protocol.

The paper is organized as follows. Section II describes the state sharing problem in asymmetric routing environments, then discusses state sharing methods between firewalls during the connection establishment procedures. In Section III, we propose an algorithm that uses M-SYN cookies by considering the state synchronization problem. The M-SYN cookie is a SYN cookie modified for multi-homed environments which includes the firewall identification number. Sections IV and V discuss performance issues and implementation results respectively, followed by the conclusion in Section VI.

II. STATE EXCHANGE FOR CONNECTION ESTABLISHMENT

This section describes the state sharing problem in MEP environments, and considers two state sharing approaches that take different steps during the TCP handshaking. Then, we briefly review the principles of a SYN cookie [3], which is applicable to state sharing among firewalls.

A. Problem Definition

We let $FW(c, s)$ denote the firewall on the one-way routing path from client host c to server host s, either c or s is in the considered AS to be protected by firewalls. We call the routing path of a connection symmetric when $FW(c, s) = FW(s, c)$. Otherwise, it is asymmetric. Suppose a multi-homed AS is with m firewalls at the entry points, i.e., $FW_i, i = 1, \cdots, m$. Without loss of generality, let FW_x denote $FW(c, s)$ and FW_y denote $FW(s, c)$. State sharing between firewalls is needed when $FW_x \neq FW_y$ for any client-server pair (c, s).

Fig. 2 shows an example of network with $m = 4$, $FW_x = FW_1$ and $FW_y = FW_2$. To do stateful inspection of a TCP connection, FW_1 needs to share the state information with

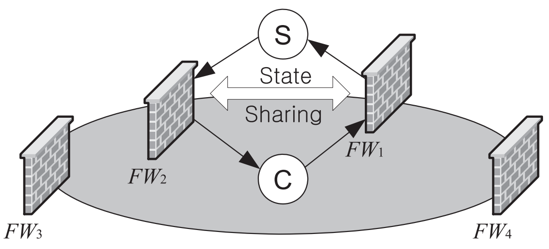


Fig. 2. State sharing problem for asymmetric routing.

FW_2. Assume that a SYN packet, referred to as SYN(c, s), passes through FW_1 and its corresponding SYN/ACK packet, referred to as SYNACK(s, c), arrives at FW_2. To check the validity of SYNACK(s, c), FW_2 needs to know the state information of this connection, otherwise it should drop the SYNACK(s, c). Thus, a state sharing mechanism is required for firewalls to protect the MEP network against possible attacks. And we should achieve it at a minimum cost.

B. Two State Sharing Methods

We can consider two possible ways of state sharing for a TCP connection according to the initiator of state sharing. One is SYN-initiated, and the other is SYN/ACK-initiated. In the SYN-initiated state sharing, if FW_x receives a SYN(c, s), it sends the connection information to FW_y. Then, FW_y inspects flows from s to c using this information. In the SYN/ACK-initiated state sharing, FW_y initiates the sharing process. Upon receiving a SYNACK(s, c), FW_y checks whether it has seen the corresponding SYN(c, s) packet before. If it has not seen, FW_y inquires other firewalls of the connection state.

B.1 SYN-Initiated State Sharing

In a SYN-initiated algorithm, a firewall receiving a SYN(c, s) initiates information exchange by sending new connection information to all other firewalls. A drawback of this method is that it is hard to know in advance which firewall will receive the SYNACK(s, c). In case of symmetric routing, state sharing among firewalls is not necessary for connections. But if a firewall cannot convince the routing path a priori, it should send the state information to every other firewall which does not need it except the designated one. Such a SYN-initiated algorithm incurs $m - 1$ extra packets per connection. Thus, messaging overhead of $m - 1$ packets per SYN packet renders firewalls vulnerable to denial of service (DoS) attacks with $O(m)$ amplification.

In order to reduce the messaging overhead, the firewall can use a table which stores routing paths for packets coming back to the AS.[1] The creation of such table is done by following the reverse process of a routing table creation because it uses returning paths instead of forwarding. To keep the table up-to-date, the firewall needs close cooperation with current routing protocols such as BGP, which is not a trivial work.

[1]In [21], a similar approach was suggested in order to check the source reachability of a traveling packet, and to filter out spoofed packets through the collaboration between ASes. However, it is inefficient for each firewall to maintain a table without support of routing protocols.

In addition to the redundant messages, a SYN-initiated approach has a race condition between state sharing and SYNACK(s, c) arrival, which is the same problem incurred by the high availability setup.

B.2 SYN/ACK-Initiated State Sharing

Under the SYN/ACK-initiated state sharing, FW_y initiates information exchange upon receiving a SYNACK(s, c). If FW_y receives a SYNACK(s, c) and it has not seen the corresponding SYN(c, s) before, i.e., $FW_y \neq FW_x$, then it requests the connection information to the other firewalls.

The advantage of this method is that extra messages are not generated for symmetric routing cases. The messaging overhead of SYN/ACK-initiated sharing is proportional to $\alpha \cdot (m - 1)$, $0 \leq \alpha \leq 1$, where α is the portion of asymmetric routing. The SYN/ACK-initiated sharing has negligible messaging overhead when $\alpha \approx 0$.

Nevertheless, it is still susceptible to DoS attacks like floods of spoofed SYN/ACK packets. Fake SYN/ACK packets amplify the number of messages because a single SYN/ACK packet invokes $m - 1$ control packets. This amplification effect causes DoS of the firewalling network.

These drawbacks motivate us to devise a novel approach without incurring the amplification effect. There are two conditions of a protocol to minimize the messaging overhead:

1. Only a valid SYN/ACK initiates state sharing,
2. FW_y inquires only FW_x instead of all the others.

In this case, the messaging overhead is only proportional to α and independent of m. To achieve this goal, upon receiving a SYNACK(s, c), the firewall needs to check the validity of the SYNACK(s, c), and identify which firewall the corresponding SYN(c, s) passed.

To design a protocol satisfying the above conditions, we manipulate the initial sequence number (ISN) of a TCP SYN packet, which is analogous to SYN cookies [3]. SYN cookies were originally suggested by Bernstein to defend a TCP server against SYN flooding attacks with IP spoofing [6]. As an extension of SYN cookies, Cascado *et al.* recently proposed a network-based flood protection scheme using the "flow cookies" [5], which is based on the same principle of host-based SYN cookies.

C. Principles of Original SYN Cookies

Let us describe the communication protocol of SYN cookies [3], which will be the basis of designing M-SYN cookies.

When a host receives a SYN packet, it stores connection information and remains in half-open state until it receives an ACK packet in reply to the SYN/ACK packet it sent. If it receives enormous forged SYN packets, i.e., SYN flooding attacks, it cannot work properly because all the resources are occupied by half-open connections. SYN cookies was designed to defend against SYN flooding attacks, whereas M-SYN cookies are designed to share connection states among firewalls. It is a particular choice of initial TCP sequence number, which represents a connection state. Instead of keeping all the connection information, hosts send out SYN/ACK packets each with the SYN cookie to achieve the scalability. Thus, hosts using

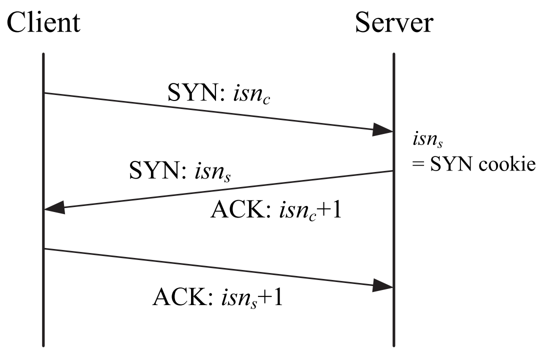


Fig. 3. TCP 3-way handshaking with SYN cookie.

SYN cookies do not have to drop connections even if the backlog queue fills up.

Fig. 3 shows a TCP 3-way handshaking with a SYN cookie. Let isn_c and isn_s denote the 32 bit ISNs sent by a client and a server, respectively. Upon receiving a SYN packet, the server generates a SYN cookie[2] according to the following formula:

$$\begin{aligned} isn_s = {} & hash(sec1, saddr, sport, daddr, dport) \\ & + isn_c + (time \times 2^{24}) \\ & + (hash(sec2, saddr, sport, daddr, dport, \\ & time) \bmod 2^{24}) + mss_{\text{index}}, \end{aligned} \quad (1)$$

where $time$ represents a 5-bit counter increasing every 64 seconds, mss_{index} represents an encoded value of the MSS in the range of 0 to 7, $sec1$ and $sec2$ represent secret keys which only the server knows, and $hash$ represents a cryptographic hash function such as MD5 or SHA-1. The other four parameters $saddr$, $sport$, $daddr$ and $dport$ represent source address, source port, destination address, and destination port of the SYN packet, respectively. If a client receives the SYN/ACK packet with the SYN cookie, i.e., isn_s, it sends an ACK packet with the acknowledge number of $isn_s + 1$. A server receiving the ACK packet checks its acknowledge number by using the following formula:

$$\begin{aligned} mss_{\text{index2}} = {} & acknum - seqnum - time \times 2^{24} \\ & - hash(sec1, saddr, sport, daddr, dport) \\ & - (hash(sec2, saddr, sport, daddr, dport, \\ & time) \bmod 2^{24}), \end{aligned} \quad (2)$$

where $acknum$ and $seqnum$ represent the acknowledge number and the sequence number of ACK packet respectively. If mss_{index2} is in the range of 0 to 7, the ACK packet is considered as legitimate, and the server creates a connection with the MSS corresponding to mss_{index2}. In case that a packet is forged, mss_{index2} tends to be very different from a value between 0 and 7.

[2]The format of M-SYN cookie that we are using is different from that of SYN cookie. Its format will be discussed in Section III-C.

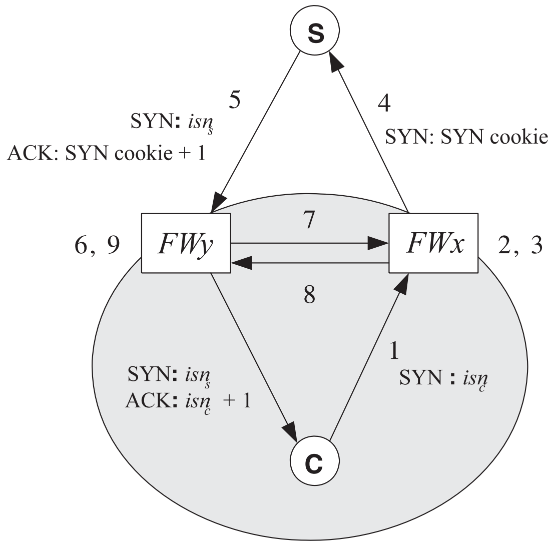


Fig. 4. Connection state exchange protocol.

III. DISTRIBUTED STATEFUL INSPECTION PROTOCOL

In this section, we propose a distributed stateful inspection protocol for coordination of multiple firewalls in an AS.

A. Protocol Design

The requirements of our protocol design for connection state exchange are as follows:

- It needs to be secure as much as a SEP firewall.
- It minimizes extra packets and computations.
- It guarantees no race condition.
- It is compatible with the current Internet.

In this protocol, we take advantage of the SYN cookie to securely exchange connection information between the two associated firewalls. While the SYN cookie manipulates SYN/ACK packets, we modify it to make it applicable to SYN packets. Unlike the SYN cookie, the M-SYN cookie uses the firewall ID instead of mss_{index} to securely record the cookie sender's information. The verification of SYN/ACK packets is performed by a keyed-hash function. To use this function, all the firewalls in the AS need to share the same secret key. Our proposed protocol is depicted in Fig. 4, and it works as follows.

1. C sends a SYN packet which arrives at FW_x.
2. FW_x examines the packet according to its packet filtering rules. If the requested connection is valid, continue the next step.
3. FW_x replaces isn_c with the M-SYN cookie, and keeps[3] the connection information[4] at the state table.
4. FW_x sends the modified SYN packet to S.
5. S sends C a SYN/ACK packet which will go through FW_y.
6. FW_y examines the SYN/ACK packet and extracts the firewall ID of FW_x. If the packet is invalid, it will be dropped. If $FW_x = FW_y$, go to step 9.
7. FW_y forwards the SYN/ACK packet to FW_x.

[3]The SYN cookie does not have this storing process.

[4]Source address, source port, destination address, destination port, and difference of the sequence numbers between isn_c and the M-SYN cookie.

8. FW_x checks the connection information of the packet and sends it to FW_y with the SYN/ACK packet. If there is no corresponding connection information, FW_x drops the packet.
9. FW_y updates its state table and replaces the acknowledge number of the SYN/ACK packet with $isn_c + 1$.
10. FW_y sends the modified SYN/ACK packet to C.

This protocol enables FW_x and FW_y to share the connection information, and forthcoming packets including the corresponding ACK can pass through the two associated firewalls directly.

Fig. 4 illustrates when C is in the AS to be protected by firewalls and S is not, but the protocol also applies to the opposite case.

B. M-SYN Cookies with TCP Reincarnation

There are regulations about TCP connection reincarnation for Internet hosts as shown in [23]. The TCP reincarnation is described as follows.

"When a connection is closed actively, it MUST *linger in TIME-WAIT state for a time* $2 \times \mathrm{MSL}$ *(maximum segment lifetime). However, it* MAY *accept a new SYN from the remote TCP to reopen the connection directly from TIME-WAIT state, if it assigns its ISN for the new connection to be larger than the largest sequence number it used on the previous connection incarnation, and returns to TIME-WAIT state if the SYN turns out to be an old duplicate."*

We build our protocol on the assumption that Internet hosts generate ISNs of larger than $SN_{\mathrm{prev}} + 32768$, where SN_{prev} denotes the sequence number used for the previous connection incarnation. We first discuss how to support fast reincarnations with SYN cookies. Then, we explain our design of M-SYN cookies with TCP reincarnations.

When S stays in TIME-WAIT state, it may accept a new SYN packet sent by C. To support the reincarnation, the ISN in the new SYN packet should be larger than SN_{prev}.

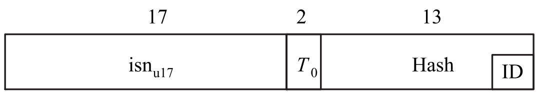
Since all arithmetic dealing with sequence numbers must be performed modulo 2^{32} [22], half of sequence numbers are larger than SN_{prev} and the others are not. So if FW_x makes a SYN cookie pseudo randomly by using a hash function without knowing SN_{prev}, the probability of the SYN cookie not being accepted by S is 1/2. Thus, we need to lower this probability.

A simple way to achieve this is to store the information about SN_{prev} during the TIME-WAIT state period. Let's denote SN_{prev} of 32 bits by $(S_{31}, S_{30}, S_{29}, ..., S_0)_2$. If the new SYN cookie keeps the first two bits of SN_{prev} and adds 2^{30} to itself, it always becomes larger than SN_{prev} in modular arithmetic, i.e.,

$$\begin{aligned}&(S_{31}, S_{30}, S_{29}, \cdots, S_0)_2 \\ &\quad < \left((S_{31}, S_{30}, *, \cdots, *)_2 + 2^{30}\right) (\mathrm{mod}\ 2^{32}) \qquad (3)\end{aligned}$$

where '$*$' is a wildcard. So if firewalls store only two bits per connection, they can support fast reincarnation exactly. However, as there are so many short-lived connections such as web traffic, it still costs a lot for firewalls to keep this information per connection for $2 \times \mathrm{MSL}$.

Another way to support fast reincarnation is to send two SYN packets with cookies that differ only in their most significant bits. In this case, S will accept only one cookie larger than SN_{prev} and reply to the firewall with its SYN/ACK packet. This method increases the overhead because of sending duplicated messages. Furthermore, if the ISN of the SYN packet is the duplication of a previous connection, then it causes to do half-open a wrong connection.



isn$_{u17}$: Upper 17 bits of isn received
T_0 : 2 LSBs of time counter, time
Hash : Hash (S_a, S_p, D_a, D_p, time, isn$_{u17}$, secret key)
ID : Firewall ID

Fig. 5. The message format of a M-SYN cookie, which is inscribed to the ISN of a SYN packet.

From the assumption for the support of fast reincarnations, hosts need to generate ISNs of larger than $SN_{\mathrm{prev}} + 32768$ to have the capability of fast reincarnation. The number 32768 comes from the following reasons. 4.4BSD-Lite adds 64000 to the ISN of each connection in addition to timer increase [29], while NetBSD-1.5.2 adds 16777216. FreeBSD-4.5 and Linux 2.4.17 use 1MHz timer so every 33 ms the ISN is increased by 32768. For OSes which use 250 KHz timer proposed in [22], it takes 132 ms to increase the ISN by 32768. However these instances are not directly related to our case of fast reincarnation because they are just concerned with increasing ISNs as a part of connection setup.

Nevertheless, fast reincarnation is made possible by increasing the ISN by 'at least' 32768 from SN_{prev}. The increment of smaller than 32768 is not enough to secure a hash field of our M-SYN cookie which will be explained next.

If the assumption is fulfilled in the TCP implementations as mentioned above, the upper 17 bits of the ISN sent by C becomes larger than SN_{prev}. Therefore, it makes sense to leave the upper 17 bits intact to support reincarnation properly.

C. M-SYN Cookie Messages

Our M-SYN cookie format is depicted in Fig. 5. It is assumed that all firewalls share a secret key and a synchronized time counter which increases every 16 seconds. The upper 17 bits of the M-SYN cookie are taken from that of the ISN of the received SYN packet. T_0 is the two least significant bits of the time counter which enables a firewall to extract the time of the cookie made. The firewall receiving the M-SYN cookie extracts the time according to the following equation:

$$\mathrm{time}_{\mathrm{input}} = \mathrm{time}_{\mathrm{curr}} + 1 - ((\mathrm{time}_{\mathrm{curr}} + 1 - T_0) \bmod 4) \quad (4)$$

where $\mathrm{time}_{\mathrm{curr}}$ is the current time. This time value, as an input of a hash function for verifying the validity of a cookie, becomes invalid after 40 seconds on average from the generation of a cookie. We will explain the meaning of (4) and time synchronization in Section IV-C.

An adversary may try to pass through the firewalls by a brute force attack such as sending SYN/ACK packets with random or sequential hashes. The M-SYN cookie uses a keyed-hash function to defend such attacks. But it has the output length of 13 bits differently from the original one of 32 bits. So the probability of randomly guessing the correct hash value in an attempt is 2^{-13}. Additionally we include the firewall ID by sacrificing the length of hash output. For instance, if an AS is with 4 firewalls, 2 bits are required to identify a firewall so that the probability of accepting a random hash increases to 2^{-11}. This implies that four out of 2^{13} fake packets can be classified as legitimate and three of them will be forwarded to other firewalls. These packets, however, will be filtered out by receiving firewalls because their state information is not in the table. The other packet will be also dropped by the first firewall due to the same reason. So the flooding attack is tolerable because it incurs only one temporary extra packet per $2^{13}/(m-1)$ attacking packets.

The firewall executes the pseudo code in Fig. 6 to generate the M-SYN cookie when it receives a SYN packet. When it receives a SYN/ACK packet, it runs the code in Fig. 7 to examine the validity of the received cookie. For simplicity, we excluded the codes for exception handling and general stateful inspection, which are not directly related to this work.

D. State Synchronization

A proposed state sharing algorithm for connection establishment is presented in the previous sections, and this section discusses further issues about stateful inspection firewalls with MEPs.

Stateful inspection firewalls should trace connection state changes. So when state change occurs, the firewall should exchange state information with the other corresponding firewall. For example, when ftp initiates a session (FTP PORT command) or when a multimedia control session opens a UDP port for streaming service, the firewall detecting the state change forwards the packet to the corresponding firewall to share the state change without race condition.

If firewalls have to check the sequence number and the acknowledge number of each packet using the cooperative stateful inspection, they should exchange the state information whenever a packet traverses. Such high overhead makes firewalls unable to be deployed in MEP environments.

Now we consider three cases that an attacker sends fake ACK packets.

Case 1) An attacker sends fake ACK packets to nonexistent connections. This attack can be used for port scanning [17], but it is ineffective under stateful inspection. When the firewall does not have the connection information in its state table, it will drop the fake ACK packets.

Case 2) An attacker sends fake ACK packets belonging to some existing connections but he/she could not sniff the connections. In this case, the attacker cannot receive the ACKs for the packets he/she sent. One probable aim of the attacker is to hinder communications by cutting off the connection, inserting corrupted data or malicious codes, and consuming host or network resources. Since the victim can usually be good at checking sequence numbers, the double-check at firewalls is redundant at normal situations. However, in terms of DoS attack defense, dropping most attacking packets through verifying sequence numbers is effective to diminish the impact of the DoS attack.

Case 3) An attacker has an ability to sniff the connection. In this case, the attacker can send spoofed packets with valid sequence numbers, and even hijack the connection. Accordingly even stateful inspection firewalls in the SEP environment cannot protect the network from this attack properly.

Since it is hard to have a perfect mechanism for checking sequence numbers, we can consider a loosely checking mechanism which is good at handling the case 2. In [25], exact checking equations were proposed. Here we propose a mechanism allowing some margin of constant or multiple window size of the TCP connection to the equations to reduce the control packet overhead.

For FIN and RST packets, the correctness of their sequence numbers must be checked accurately to avoid erasing valid state information from the state table. In this case, it is better for the firewall to loosely check the sequence numbers first and then updates state information by exchanging control packets and checks them again tightly.

On the Internet, the majority of TCP sessions do not experience route changes during their connection lifetime [20]. However some connections undergo route oscillations. If a packet arrives at a firewall which does not have its connection information, it would be dropped. Therefore, to run under route oscillations as desired, other firewalls that have not participated in connection establishment also should share minimum information of the connection. But route oscillations do not happen frequently and it is desired to protect long connections that tend to be important. So if firewalls share minimum connection informations at long intervals, they can tolerate route changes with low overhead.

IV. PERFORMANCE

In this section, performance issues are discussed in terms of processing overhead and messaging overhead. As well, we briefly discuss security and scalability of the algorithm.

A. Processing Overhead

Since the algorithm does not contain loops or recursions, the time complexity is $O(1)$ for processing a single SYN or SYN/ACK packet. Nevertheless, there are two computation intensive parts for further considerations: a hash function and a code for sequence number translation.

A.1 Hash Function

A hash function is called twice for each connection. One is for generating a cookie and the other is for verifying the cookie. In case of flooding SYN/ACK packets, either directly from attacking sources or indirectly from reflectors [19], firewalls have to call the hash function as much. Thus, the hash function gives significant impact on the overall performance so that it must be chosen carefully. There exist commercial network security processors that achieve multi-gigabit throughput using MD5 and SHA-1 hash functions. Also, fast hash functions like UMAC and MMH can achieve multi-gigabit performance even on a 350 MHz Pentium II PC [4].

```
If a SYN packet is received and allowed in the filtering rule then
    cookie = ( isn_c & 0xffff8000 ) + ( time%4 << 13 )
          + HASH(secret, sa, sp, da, dp, time, isn_c>>15) % 2^13 + FWid
    SNdiff = cookie - isn_c
    Update StateTable(Store sa, sp, da, dp, time, SNdiff)
    Change TCPseqnum to cookie
    Recalculate TCPchecksum
    Send the packet
Endif
```

Fig. 6. Generation of a cookie for a SYN packet.

```
If a SYN/ACK packet is received then
    temp_cookie = TCPacknum - 1
    temp_time = time + 1 - ( time + 1 - ( temp_cookie >> 13 ) ) % 4
    id = ( temp_cookie - HASH(secret, da, dp, sa, sp, temp_time, temp_cookie>>15) ) % 2^13
    If id == FWid
        If the session is in the StateTable
            Update StateTable
            Change TCPacknum to TCPacknum-SNdiff
            Recalculate TCPchecksum
            Send the packet
        Else
            Drop the packet
        Endif
    Else If id>=0 and id<=MAXid
        Forward the packet to FW(id)
    Else
        Drop the packet
    Endif
Endif
```

(a)

```
If a SYN/ACK packet is forwarded from FW(id)
    If the session is in the StateTable
        Update StateTable
        Send the connection information and the packet to FW(id)
    Else
        Drop the packet
    Endif
Endif
```

(b)

Fig. 7. Verification of the cookie from a SYN/ACK packet; (a) FWy–SYN/ACK received (Section III-A step 6, 7) and (b) FWx–SYN/ACK forwarded(Section III-A step 8).

A.2 Sequence Number Translation

In order to change isn_c, a firewall needs to translate all the sequence numbers of packets in one direction and all the acknowledge numbers in the opposite direction, and recalculate their TCP checksums. The sequence number (SN) translation can be done with one addition or one subtraction. TCP checksum (CS) can be recalculated by the following arithmetic [24]:

$$CS_{new} =\sim (\sim CS_{\text{old}} + \sim SN_{\text{old}} + SN_{\text{new}}), \tag{5}$$

where $\sim a$ means one's complement of a. This implies that the sequence number translation gives negligible impact on the performance.

B. Control Packet Overhead

The proposed protocol uses the minimum number of extra packets for each connection establishment. That means two packets used for the request-and-reply state exchange when the routing path is asymmetric. Furthermore, relaxed checking of sequence numbers can further reduce the control packets. Also as each control packet requires some processing overhead including authentication, we can put multiple queries not in urgent need into one packet.

C. Security

Firewalls with MEPs need to be as secure as SEP firewalls. We consider two types of attacks in view of security depending on how the firewall reacts when it receives SYN and SYN/ACK packets.

C.1 DoS Attacks

There are two possible DoS attacks: SYN flooding and SYN/ACK flooding. On receiving a SYN or SYN/ACK packet, the firewall needs to execute the hash function. As explained in Section III-C, such attacks cause only one extra control packet per $2^{13}/(m-1)$ attacking packets and most attacking packets are dropped immediately.

C.2 Replay Attacks

In order to prevent replay attacks, the current time is used as an input of the hash function. After 64 seconds, an M-SYN cookie becomes invalid and the packet will be dropped accordingly. The proposed protocol prevents an attacker from guessing the M-SYN cookie without having seen it recently, which is the same principle of the SYN cookie. To prevent the replay attack, it is assumed that all firewalls have a synchronized time counter, which increases every 16 seconds. We inserted the two least significant bits T_0 of the time counter, $\mathrm{time}_{\mathrm{org}}$, and extracted $\mathrm{time}_{\mathrm{input}}$ by using (4). Then, the $\mathrm{time}_{\mathrm{input}}$ has the following property.

$$\begin{aligned}&\mathrm{time}_{\mathrm{org}} = \mathrm{time}_{\mathrm{input}}\\ \Longleftrightarrow\ &\mathrm{time}_{\mathrm{org}} - 1 \le \mathrm{time}_{\mathrm{curr}} \le \mathrm{time}_{\mathrm{org}} + 2. \qquad (6)\end{aligned}$$

The reason we allow $(\mathrm{time}_{\mathrm{org}} - 1)$ and $(\mathrm{time}_{\mathrm{org}} + 2)$ is to tolerate time asynchronization between firewalls for up to 16 seconds. If the time asynchronization period does not exceed 16 seconds, the cookie is valid for 16 to 64 seconds, which equals to 40 seconds on average.

D. Scalability

A good feature of our proposed protocol is its scalability because it runs independently of the number of firewalls in an AS. This is caused by the fact that a connection exchange occurs just between the two associated firewalls with their own identifiers. Our protocol has low message complexity and does not induce amplified messages. If one packet triggers m extra messages, it will be very much susceptible to DoS attacks. Fortunately, M-SYN cookies generate at most one control packet per SYN/ACK packet. Desirably it does not generate any unnecessary control packets for symmetric connections, thus the scalability is achieved regarding the number of exchanged packets for state validation.

For storing the firewall ID in a M-SYN cookie, more space is required for more firewalls. But allowing one more bit to the firewall ID field doubles the number of partaking firewalls. Furthermore, we do not loose the protection power of M-SYN cookies for a reasonable range of firewall numbers. For instance, even with 16 firewalls, only one of 546 packets can pass through firewalls, but the connection will be dropped immediately, which is discussed in Section III.C. From the facts discussed above, the scalability can be achievable at the expense of hash space.

V. IMPLEMENTATION

We have implemented the proposed protocol using the netfilter in the Linux kernel.[5] This section starts with a brief introduction of the Linux network kernel and the netfilter, and then discusses our implementation. For an experimental setup, we construct a network consisting of three firewalls.

A. Linux Network Kernel and Netfilter

When the Linux kernel receives an IP packet, it calls ip_rcv() function. ip_rcv() checks the validity of the packet and calls

[5]Linux kernel version 2.4.19 is used for implementation.

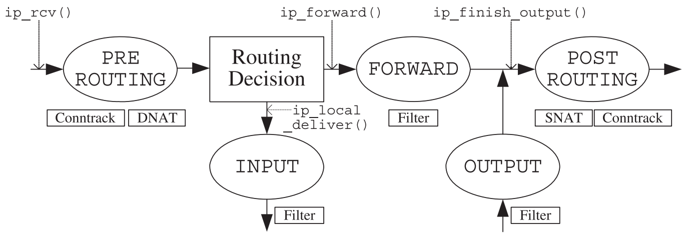


Fig. 8. Netfilter architecture.

ip_route_input() to examine whether the destination is local or not. If the destination is local, i.e., the host itself, the packet is delivered to ip_local_deliver(). Otherwise the packet traverses ip_forward() and ip_finish_output() and finally goes out to another network.

The netfilter is the framework in the Linux 2.4.x kernel for packet filtering, network address translation (NAT) and other packet manipulation. It is a set of hooks in the network stack of Linux kernel which allows kernel modules to register callback functions that are called whenever a packet traverses one of these hooks [15]. The hooks are put at between the functions mentioned above.

Fig. 8 shows the architecture of the netfilter [16]. Five ellipses represent the hooks of netfilter and small rectangles are functions that are invoked when packets encounter hooks. Besides static packet filtering, the filter provides stateful inspection functionalities. DNAT and SNAT represent destination and source NATs respectively, and perform address translations of connections. DNAT applies to port forwarding, transparent proxying and so on. SNAT changes the source address like the IP masquerading feature of Linux. Conntrack tracks connections, and supports NAT and stateful inspection filtering.

B. Protocol Implementation

We implemented our protocol as a netfilter module. We installed our function SCGEN between SNAT and Conntrack at POSTROUTING, and SCCHECK between Conntrack and DNAT at PREROUTING. SCGEN generates M-SYN cookies for new connections by the codes in Fig. 6 and SCCHECK checks the validity of received M-SYN cookies by the codes in Fig. 7.

The main factor in deciding the location of each function block is related to the use of NAT in the MEP network. For example, if we turn on SNAT, FW_x changes the source address of the outgoing SYN packet, so the returning SYN/ACK packet has the changed destination address. Assuming an MEP environment and the routing path is asymmetric, FW_y does not know how to restore the original destination address, especially if SNAT is dynamic. It is exactly the same problem as observed in our state sharing problem, so it can be rightly solvable by our protocol. After FW_y validates the M-SYN cookie, it requests connection information to FW_x. Then FW_x responds to FW_y with state information including NAT information. Using this information, FW_y performs NAT. Because FW_y validates the cookie by the changed address, FW_x has to generate the cookie with the changed address. Therefore SCGEN needs to be called after SNAT.

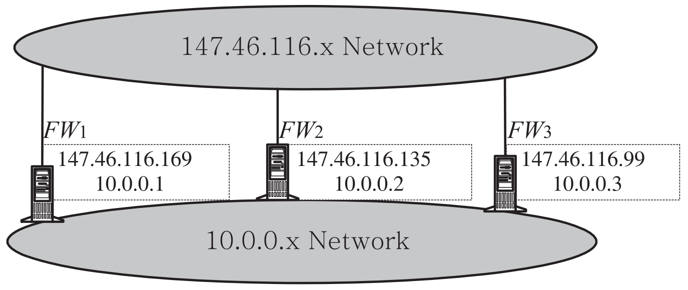


Fig. 9. Testbed configuration.

C. Experimental Results

We conducted experiments for our proposed protocol on the testbed shown in Fig. 9 which consists of three firewalls. Each PC firewall is equipped with an Intel Pentium 4 1.8 GHz CPU and two 100 Mbps Ethernet cards.

C.1 Protocol Validation

Following the initial design purpose, we tested our protocol with asymmetric routing paths by allocating a designated firewall for each direction of flow. Along with the experiments, we verified that our protocol provides the intended functions properly under various connection scenarios. These experiments covered the case of time asynchronization among firewalls, and verified the validity of the considered algorithms including (4) and (6).

C.2 Hash Functions

We experimented with various hash functions to investigate their effects on performance. Every SYN and SYN/ACK packet arriving at the firewall calls for the hash function. We measured the packet drop rate with the increase of connection requests, which is shown in Fig. 10. Here, the packet drop rate implies the dropping percentage of new TCP connections. NO HASH stands for the case when the null function is used. When SHA-1 is used, packets starts to be dropped at 65 Kpps. In case of MD5, the packet drop rate increases drastically from 75 Kpps. For UMAC [4], packets begins to be dropped at near 82 Kpps, which is the similar performance to the case of NO HASH. In this experiment, extra processing overhead results in 20% $\sim$ 0% performance degradation depending on the hash function. It implies that a fast hash function does not cause too much performance degradation. Thus, the proposed protocol constituted by hash functions can be used efficiently for protecting a network with MEP.

C.3 Fake SYN/ACK Attack

The effectiveness of the proposed protocol with respect to verifying SYN/ACK packets is measured under a flooding attack of fake SYN/ACK packets. We generated 1000000 SYN/ACK packets with random acknowledge numbers. Among 1M forged packets, only 251 packets, which are very close to the expected value of 244,[6] were forwarded to the other firewalls, and they were all dropped at the receiving firewalls. This verifies that no fake SYN/ACK packets could succeed to penetrate into the network.

[6]In case of $m = 3$, the expected value is $1000000 \times (3-1)/2^{13} \approx 244$.

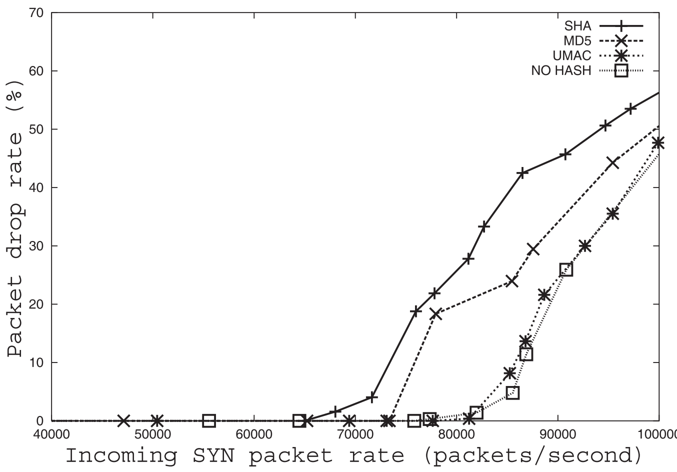


Fig. 10. Hash function test.

VI. CONCLUSION

The state sharing problem occurs among firewalls in the MEP network as the Internet intrinsically allows connections to experience asymmetric routing paths. Conventional firewalls have topological restrictions since they have no ability to do stateful inspection in MEP environments. Such a problem has hindered deploying stateful inspection firewalls in the MEP environments.

In this paper, we described the state sharing problem between two associated firewalls in the MEP network. Also we proposed a distributed stateful inspection protocol which exchanges connection information and checks its validity by using our proposed M-SYN cookie. Differently from the SYN cookie, our cookie contains the field of firewall ID to indicate which firewall the SYN packet passed through, and a hash value to check the validity of a connection request. When the firewall receives a SYN/ACK packet, it examines the hash value and extracts the firewall ID from the acknowledge number to defend against DoS attacks employing fake SYN/ACK flooding.

The proposed protocol is scalable because the control messages are exchanged between the two associated firewalls of an asymmetric connection, regardless of the number of entry points in an AS. The protocol requires low processing and messaging overhead, thereby it can be applicable to current Internet environments. For future work, we left the case of supporting UDP sessions in the MEP network which are connectionless.

REFERENCES

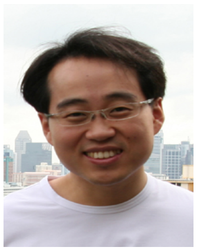
[1] A. Akella, A. Shaikh, and R. Sitaraman, “A measurement-based analysis of multihoming,” in *Proc. ACM SIGCOMM*, 2003.

[2] S. Bellovin, *Distributed Firewalls; login: Magazine*, special issue on security, Nov. 1999.

[3] D. J. Bernstein, SYN Cookies Homepage, 1996. [Online]. Available: http://cr.yp.to/syncookies.html.

[4] J. Black, S. Halevi, H. Krawczyk, T. Krovetz, and P. Rogaway, “UMAC: Fast and secure message authentication,” in *Proc. Advances in Cryptology–CRYPTO*, 1999.

[5] M. Casado, A. Akella, P. Cao, N. Provos, and S. Shenker, "Cookies along trust-boundaries (CAT): Accurate and deployable flood protection," *Usenix SRUTI'06: 2nd Workshop on Steps to Reducing Unwanted Traffic on the Internet*, 2006.
[6] CERT/CC, "TCP SYN flooding and IP spoofing attacks," CERT Advisory CA-1996-21, Sept. 1996.
[7] Check Point Software Technologies Ltd. (Aug. 2005). Stateful Inspection Technology. Check Point Tech Note. [Online]. Available: http://checkpoint.com/products/downloads/Stateful_Inspection.pdf.
[8] J. Han, D. Watson, and F. Jahanian, "An experimental study of Internet path diversity," *IEEE Trans. Dependable and Secure Computing*, vol. 3, no. 4, pp.273–288, Oct.–Dec. 2006.
[9] Y. He, M. Faloutsos, and S. Krishnamurthy, "Quantifying routing asymmetry in the Internet at the AS level," in *Proc. IEEE GLOBECOM*, 2004.
[10] Y. He, M. Faloutsos, S. Krishnamurthy, and B. Huffaker, "On routing asymmetry in the Internet," in *Proc. IEEE GLOBECOM*, 2005.
[11] S. Ioannidis, A. Keromytis, S. Bellovin, and J. Smith, "Implementing a distributed firewall," in *Proc. ACM CCS*, 2000.
[12] J. Johnson. (June 2002). BGP Is A Reachability Protocol. A NANOG Presentation. [Online]. Available: http://www.nanog.org/mtg-0206/ppt/jerm2/.
[13] J. Kim, S. Bahk, and H. Lee, "A connection management protocol for stateful inspection firewalls in multi-homed networks," in *Proc. IEEE ICC*, June 2004.
[14] Stonesoft. (Oct. 2001). Multi-Link Technology. [Online]. Available: http://www.stonesoft.com/products/whitepapers.
[15] Netfilter Homepage. [Online]. Available: http://www.netfilter.org.
[16] R. Russel and H. Welte, *Linux netfilter Hacking HOWTO*, June 2002.
[17] Nmap Homepage. [Online]. Available: http://www.insecure.org/nmap.
[18] V. Paxson, "End-to-end routing behavior in the Internet," in *Proc. ACM SIGCOMM,* 1996.
[19] V. Paxson, "An analysis of using reflectors for distributed denial-of-service attacks," *Computer Communications Review 31 (3)*, July 2000.
[20] K. Park and H. Lee, "On the effectiveness of probabilistic packet marking for IP traceback under denial of service attack," in *Proc. IEEE INFOCOM*, Apr. 2001, pp.338–347.
[21] K. Park and H. Lee, "On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law internets," in *Proc. ACM SIGCOMM*, Aug. 2001, pp.15–26.
[22] J. Postel, *Transmission Control Protocol*, STD 7, RFC 793, Sept. 1981.
[23] R. Braden, "Requirements for Internet hosts–communication layers," STD 3, RFC 1122, Oct. 1989.
[24] A. Rijsinghani, "Computation of the Internet checksum via incremental update," RFC 1624, May 1994.
[25] G. Rooij, "Real stateful TCP packet filtering in IP filter," 10th USENIX Security Symposium invited talk, Aug. 2001.
[26] Q. Vohra and E. Chen, "BGP support for four-octet AS number space," Work in progress, Internet Draft draft-ietf-idr-as4bytes-13.txt, Feb. 2007.
[27] D. Vukadinovic, P. Huang, and T. Erlebach, "A spectral analysis of the Internet topology," Technical Report ETH-TIK-NR 118, 2001.
[28] D. Welch-Abernathy, *Essential Check Point FireWall-1*, Addison-Wesley Publishers, Jan. 2002.
[29] G. Wright and W. Stevens, *TCP/IP Illustrated, Volume 2: The Implementation*, Addison-Wesley, 1995.



Jin-Ho Kim received the B.S. degree in electrical engineering from Seoul National University in 1997, and the M.S. and Ph.D. degrees in electrical engineering and computer science from Seoul National University in 1999 and 2004, respectively. He is a software engineer at Google. Prior to joining Google, he was a senior engineer at Samsung Electronics from 2004 to 2007. His research interests include internet protocols, architecture, and network security. He is a member of the IEEE.

Heejo Lee is currently an Associate Professor at the Division of Computer & Communication Engineering, Korea University, Seoul, Korea. Before joining Korea University, he was at AhnLab, Inc. as a CTO from 2001 to 2003. From 2000 to 2001, he was a postdoc at the Department of Computer Sciences and the security center CERIAS, Purdue University. He received his B.S., M.S., and Ph.D. degrees in Computer Science and Engineering from POSTECH, Pohang, Korea. Dr. Lee serves as an editor of Journal of Communications and Networks. He has been an advisory member of Korea Information Security Agency and Korea Supreme Prosecutor's Office. As well, he served as an advisor for constructing the Nat'l CERT in the Philippines (2006), the consultation of Cyber Security in Uzbekistan (2007).

Saewoong Bahk received B.S. and M.S. degrees in Electrical Engineering from Seoul National University in 1984 and 1986, respectively, and the Ph.D. degree from the University of Pennsylvania in 1991. From 1991 through 1994 he was with AT&T Bell Laboratories as a member of technical staff where he worked for AT&T network management. In 1994, he joined the school of electrical engineering at Seoul National University and currently serves as a Professor. He has been serving as TPC members for various conferences including IEEE ICC, GLOBECOM, INFOCOM, PIMRC, WCNC, etc. He is on the editorial board of Journal of Communications and Networks (JCN) and editor-in-chief of KICS Journal. His areas of interests include performance analysis of communication networks and network security. He is an IEEE Senior Member and a Member of Whos Who Professional in Science and Engineering.