TECHNICAL REPORT 05-01

Bidirectional Scan combined with Packet Signature for TCP Stream Level Data Inspection

Yutaka Sugawara and Kei Hiraki

January 26, 2004

DEPARTMENT OF INFORMATION SCIENCE
FACULTY OF SCIENCE, UNIVERSITY OF TOKYO

7–3–1 HONGO, BUNKYO-KU TOKYO, 113 JAPAN

TECHNICAL REPORT 05-01

TITLE
Bidirectional Scan combined with Packet Signature for TCP Stream Level Data Inspection

AUTHORS
Yutaka Sugawara and Kei Hiraki

KEY WORDS AND PHRASES
pattern match, FPGA, TCP stream, state-holding attack

ABSTRACT
In this paper, we propose techniques to realize high-speed hardware based TCP-stream-level pattern match unit for NIDS while increasing the tolerance against state-holding attacks. Since state of each stream is necessary for TCP-stream-level matching, it would be a target of state-holding attacks. By combining the SBT method [1] with bi-directional scanning technique, packet loss and reorder can be managed with small amount of memory. Using packet signature, virtually strict retransmission consistency check can be done while reducing the memory usage. As a result, the tolerance against state-holding attacks is increased.
We implemented a TCP stream level pattern match mechanism using SBT method, bi-directional scanning, and packet signature. We used a 1000 characters rule set, and evaluated the case when slow paths are not executed. In the evaluation, we found that 13.1Gbps is achieved when there are no packet drop and reorder, and 11.2Gbps when there are packet drops and reorders. In addition, it is shown that our system provides 30–100 times large window size for each connection compared with a non-optimized case when the system received a state-holding attack.

REPORT DATE January 26, 2004	**WRITTEN LANGUAGE** English
TOTAL NO. OF PAGES 21	**NO. OF REFERENCES** 18

ANY OTHER IDENTIFYING INFORMATION OF THIS REPORT

DISTRIBUTION STATEMENT
This technical report is available ONLY via anonymous FTP from `ftp.is.s.u-tokyo.ac.jp` (directory `/pub/tech-reports`).

SUPPLEMENTARY NOTES

DEPARTMENT OF INFORMATION SCIENCE
Faculty of Science, University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo 113, Japan

Bidirectional Scan combined with Packet Signature for TCP Stream Level Data Inspection

Yutaka Sugawara and Kei Hiraki

1 Introduction

Recently, the Internet has grown rapidly, and become one of the most important infrastructures of our life. Therefore, keeping a correct operation of the Internet is an important task. A factor that prevents a normal operation of the Internet is security threat such as attack and information leakage. As the Internet is growing, damages by security threats are increasing rapidly. Therefore, keeping the security of the Internet is our duty.

Intrusion detection system (IDS) [2] [3] [4] is a widely used way to contain security threats. IDS is a system that detects security threat such as attack and information leakage. IDS records discovered security threats, or even shutdown the packets to interrupt the threats. IDS is categorized into network based IDS (NIDS), host based IDS (HIDS), and combination of them. NIDS detects security threats by observing all the packets in a target subnet. On the other hand, HIDS is deployed in the target hosts themselves, and detects security threats by observing the status of the target hosts. Since HIDS has problem in working amount of administrators and the operation when the host is compromised, NIDS has to be deployed even when HIDS is used. Therefore, NIDS is mandatory; thus we focus on NIDS in this paper.

In NIDS, The methods to analyze network data is roughly divided into signature-based methods [2] [5] [6] and anomaly-based methods [7] [8]. In the signature-based methods, an NIDS discovers security threats by finding particular octet strings contained in the packet data in conjunction with the header information. In the anomaly-based methods, on the other hand, an NIDS indirectly discovers security threats by detecting abnormal behaviors of networks and target hosts. Since a reliable solution to find all the attacks is not yet established for anomaly based method, part of attacks will be overlooked, allowing evasion vulnerability. On the other hand, using signature based method, registered attacks are reliably discovered. Therefore, to avoid evasion vulnerability, signature based method is mandatory.

1.1 TCP-stream-level pattern matching

The process of signature based NIDS is mainly divided into packet header check and octet pattern search in payload data. The octet pattern search is further divided into exact string match and regular expression match. The pattern match is performed either in per-packet basis or per-TCP-stream basis.

In the former method, the pattern match is applied to just each packet. Therefore, when the target pattern spans multiple packets, the pattern cannot be discovered, as shown in Figure 1. This is evasion vulnerability because attackers can evade the NIDS by splitting the target pattern into multiple packets. On the other hand, in per-TCP-stream basis, all the octet patterns can be discovered regardless of the pattern fragmentation [1].

[1] Similar discussion stands for IP packet fragmentation [9]. Actually, there are tools to split IP packets into multiple fragments such as Fragrouter [10]. Since the problem can be solved using the same way as the fragmented TCP stream problem, we omit the discussion for fragmented IP packets.

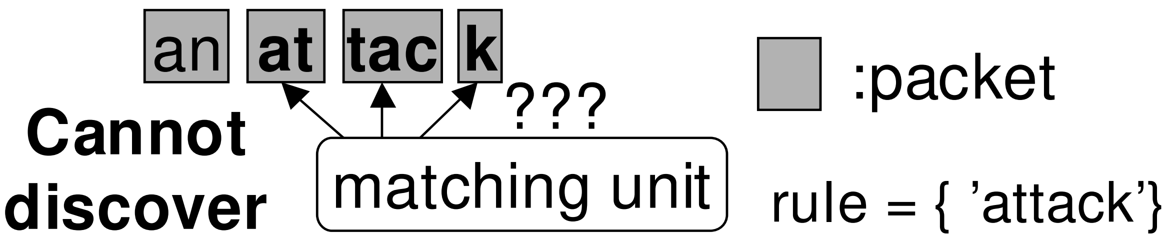

Figure 1: Evasion caused by fragmented pattern

In TCP stream pattern match, it is necessary that each stream be recovered from the packets. Note that the stream octet sequence itself is not necessary as long as the match result is correct. In addition, since logically same octet in a sequence space may be sent multiple times in TCP, retransmission consistency check [11] is necessary to prevent evasions.

1.1.1 Coping with state-holding attacks

Since state of each connection is necessary for TCP stream pattern match for NIDS, it become a target of state-holding attacks [11] [12]. The state-holding attack is a DoS attack by forcing an NIDS to consume all the memory necessary for holding state of each connection. When the NIDS run out of the memory, there are two decisions [12]; give up to check the packets (fail-open), or discard packets that cannot be checked (fail-close). First decision is to give up the check of packets for which the memory resource cannot be allocated. When this method is used, part of the attacks will be overlooked by the NIDS. Therefore, the correctness of the NIDS is not guaranteed. Second decision is to discard a part of connection data and discard the TCP packets whose connection data is not found in the NIDS memory. Using this method, though part of the innocent connections would be aborted, all the packets can be checked, preserving the correctness. Since the purpose of the state-holding attack is to hide following attacks from NIDS, the correctness should be preserved. Therefore, we discuss the second case. Nevertheless, since the network communication is harmed by the memory overflow, we must try to avoid or minimize the memory overflow.

Since the memory resources of hosts in the site limit the number of connections and the amount of unacknowledged data, NIDS memory overflow can be minimized or avoided by increasing the NIDS memory size. However, the memory size cannot be unlimitedly increased. Another way to mitigate the memory resource overflow problem is to reduce the amount of memory required for connection states and unacknowledged data.

1.2 Proposal

To apply the NIDS to recent high-speed networks such as 10Gbit Ethernet and OC-192, hardware based method is essential. In addition, the state size for each connection must be minimized to realize fast switch between connections and minimize state-holding attacks. We proposed SBT method, a hardware based exact string matching method that reqlize both high-speed and small state size. However, the memry usage cannot be reduced just using the SBT method, as we describe in Section 2.

When existing methods are used for stream recovery, all out-of-order packets are discarded [13], or all the unacknowledged data is buffered [14]. To minimize memory usage without minimizing packet discarsion, another method is necessary. Using bi-directional scanning technique proposed in this paper, the goal is realized.

In addition, to the best of our knowledge, there is no study about memory usage optimization for retransmission consistency check. In a naive implementation, all unacknowledged data have to be saved, increasing the memory usage. In this paper, we propose packet signature method to enable a *virtually* strict retransmission consistency check while reducing the memory usage.

In Section 2, we characterize the memory usage for TCP-stream-level pattern match and make strategies to reduce the usage. Section 3 describes the bi-directional scanning technique. Packet signature method is presented in Section 4. In Section 5, we evaluate an FPGA based TCP stream pattern match system with bi-directional scanning and packet signature. Section 6 presents related works, and we conclude this paper in Section 7.

2 Reducing Memory Usage for TCP-stream-level Matching

In TCP-stream-level pattern match, the following procedures are necessary that are not required in packet-level pattern match.

Stream Recovery Since each TCP stream is sent using multiple packets, pattern match unit must recover the TCP stream from the packets, and apply the pattern match to the recovered stream. Note that it is not necessary that the octet sequence of each stream be provided in a serialized form, ss long as the matching unit returns a correct answer.

Retransmission Consistency Check Because of retransmissions of TCP, an octet in a sequence space may be sent multiple times. If different octets are sent in the multiple retransmissions [2], the consistency of the pattern match is lost. Therefore, attackers can evade the NIDS by retransmitting different data. To avoid the vulnerability, the pattern match unit must check that the retransmission packet contains the same data as the first data.

Both of the procedure use memory, being a target of state-holding attack. The security can be guaranteed by fail-close strategy [12]; i. e. discarding packets that cannot be checked by NIDS because of insufficient resource. However, it forces a part of innocent connections to be halted, resulting a DoS attack. Therefore, the memory resource usage must be minimized to avoid or minimize the state-holding attack.

2.1 Memory Usage Characterization

To discuss memory reduction strategies, we characterize the memory usage of the stream recovery and the retransmission consistency check.

Stream Recovery In TCP-stream-level string matching, the target patterns that span multiple packets must be discovered. To discover such fragmented patterns correctly, information of first and last $l-1$ octets of each unacknowledged fragment must be saved, where l is the length of the longest rule string. The situation is shown in Figure 2.

Note that it is not necessary that the octets themselves be saved, as long as attackers cannot evade the scanning unit. For example, in Figure 2, when the fragment 1 is scanned, the internal state bits of the matching unit can be saved instead of the octets X. When the gap 2 is filled, the scan is possible by restoring the saved state onto the matching unit.

[2]Though this is implicitly prohibited in RFC793 [15], attackers may not obey the standard.

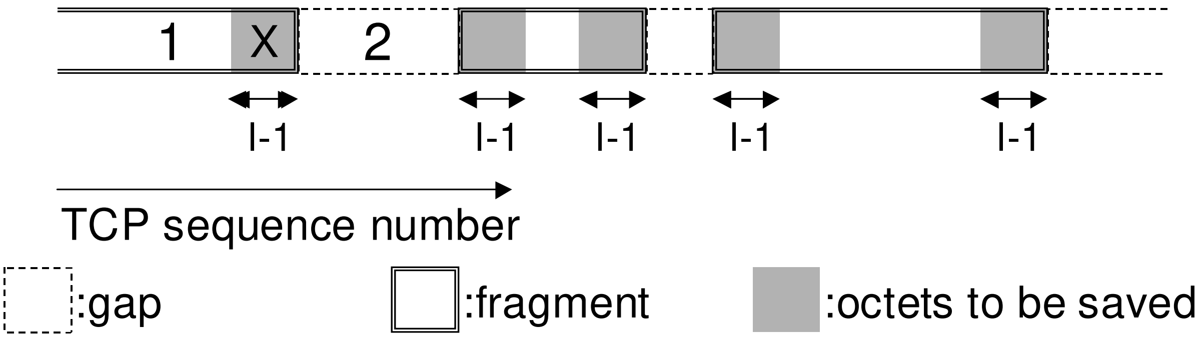

Figure 2: Saving information of octets at the edges of fragments

In a normal case, the number of holes in the sequence space is small. However, when an attacker crafts the transmission, there would be many sequence holes. Therefore, memory usage for each fragment edge data must be minimized to avoid or minimize memory overflow. When no optimization is performed, the octets at each fragment themselves are saved.

Retransmission Consistency Check For retransmission consistency check of a packet, the information of the first transmission data is necessary. To strictly check that the retransmitted data is identical to the first one, all the octets of first transmission must be saved. Therefore, the necessary memory size is total size of unacknowledged data. If a strict check is required, no memory usage optimization is possible. However, *virtually* strict check is possible while reducing the memory usage.

2.2 Memory Reduction Strategy

Stream Recovery To reduce the memory usage for the stream recovery procedure, it is necessary to compress information of octets at fragment edges. Our idea to compress the information is to save the internal state bits of the matching unit instead of the last $l-1$ octets of each fragment. When the number of state bits is smaller than $8(l-1)$, the memory usage can be eliminated. However, when existing hardware-based string matching algorithm is used, many state bits are used. Therefore, to enable this optimization, another hardware-based algorithm is necessary that uses small number of state bits.

In addition to compressing the last $l-1$ octets of each fragment, it is also necessary that first $l-1$ octets are compressed. The last $l-1$ octets cannot be compressed because the octets are scanned in increasing order of sequence number. Therefore, scanning octets in both forward and backward directions, the first $l-1$ octets can be compressed. Note that above discussion does not stand for small packets. Therefore, small packets must be processed in a special way, as described in Section 3.

Retransmission Consistency Check To reduce the memory usage for retransmission consistency check, part of packet data must be discarded. Since the information of the first packet is not complete, equality comparison procedure would return a wrong answer. However, it is possible to limit the probability of mistake to almost zero, and prevent attackers from arbitrarily cause the procedure to return a wrong answer. When part of the first retransmission data information must be discarded, it is the best thing we can do.

SN_b	Sequence number of the starting octet of the fragment
SN_e	Sequence number plus 1 of the ending octet of the fragment
s_b	Ending state of backward scan at the starting edge of the fragment
s_e	Ending state of forward scan at the ending edge of the fragment

Table 1: Information stored for each fragment

Specifically, when the original data size is s_o, and the number of bits allowed to store the information is s_s, theoretically minimum probability of the mistake is $2^{-s_s} - 2^{-s_e}$. By using s_o bits of hash value calculated by random functions such as MD5 or AES, this theoretical limit is achieved. To prevent attackers from arbitrarily forcing the procedure to return a wrong answer, the random functions must be secret.

The packet signature in Section 4 is based on this idea. In the method, the packet boundaries must be taken into account, as described in Section 4.

3 Bi-directional Scanning

3.1 Adding Backward Scan

Using SBT method [1], number of state bits is reduced. As a result, small amount of memory is necessary to store the information of last $l-1$ octets of each fragment. However, $8(l-1)$ bits of memory is still necessary to store first $l-1$ octets of each fragment.

The first $l-1$ octets cannot be compressed because the octets are feed to the matching unit in an increasing order of sequence number. Therefore, when the octets are scanned in backward direction (i.e. decreasing order of sequence number), first $l-1$ octets can be compressed.

Note that this discussion does not stand when a target pattern spans more than 3 packets. This happens just when payload size is shorter than $l-1$ octets. In this case, the packet must be handled in a special way. First we describe an algorithm to process packets with payloads of more than $l-1$ octets, and discuss how to handle small packets later.

3.2 Large Packet Reception Procedure

3.2.1 Fragment information management

To find all the patterns, ending search states at the edges of each fragment have to be saved. To find the appropriate edge state for a new packet, the range of each fragment in the sequence space is also necessary. Therefore, we use a set of four variables (SN_b, SN_e, s_b, s_e) for each fragment; each variable being described in Table 1.

3.2.2 Processing a new packet

When a new packet has arrived, processing strategy must be changed in the following respects. Note that more than one of these conditions can be satisfied simultaneously.

1. Whether the preceding fragment has been already received or not.
   When the preceding fragment has been already received, the ending state of the forward scan at the end of the fragment is available. Therefore, continuing the forward scan starting from the saved state, we can find all the patterns that span the fragment and the new packet.

Name	Satisfied conditions
PREC	1.
SUCC	2.
CONNECT	1., 2., and 3.
NEW	4.

Table 2: Possible combinations of satisfied conditions

On the other hand, when the preceding fragment has not been received yet, divided patterns must be discovered by backward scan. Thus, backward scan is performed on the new packet, and the ending state is saved. The saved state is used for backward scan when the preceding fragment is arrived. As a result, divided patterns are discovered correctly.

2. Whether the succeeding fragment has been already received or not.
   This point is similar to 1; thus, when the fragment has been already received, a backward scan is performed from the saved state at the starting point of the fragment. When the fragment has not been received, the forward scan result state of the packet is saved for later use.

3. Whether two fragments are connected by the new packet or not.
   When the new packet connects two fragments, the fragment information have to be merged into one.

4. Whether a new fragment is produced or not .
   When the new packet produces a new fragment, new fragment information has to be added.

Table 2 shows all the possible combinations of satisfied conditions. We name each combination as shown in the table. Based on the satisfied conditions, the procedure for each case is determined.

PREC In this case, just the condition 1. is satisfied. Therefore, forward scan has to be performed for the new packet, starting from the saved state at the end of the preceding fragment. In addition, since condition 2 is not satisfied, the ending state of the forward scan has to be saved for later use. This case is shown in Figure 3

SUCC In this case, just the condition 2 is satisfied. Therefore, backward scan has to be performed for the new packet, starting from the saved state at the beginning of the succeeding fragment. In addition, since condition 1 is not satisfied, the ending state of the backward scan has to be saved for later use. This case is shown in Figure 4

CONNECT In this case, condition 1, 2, and 3 are satisfied. All the procedure for the conditions must be performed. To handle condition 1., forward scan is performed for the new packet, starting from the saved state at the end of the preceding fragment. Then, to handle condition 2, backward scan is performed for the new packet, starting from the saved state at the beginning of the succeeding fragment. Finally, since 3 is satisfied, the fragment information of the two existing fragments are merged into one. This case is shown in Figure 5

NEW In this case, just the condition 4 is satisfied. Therefore, new fragment information is allocated. In addition, since neither condition 2 nor condition 3 is satisfied, the ending states of scans of both the directions are saved for later use. This case is shown in Figure 6

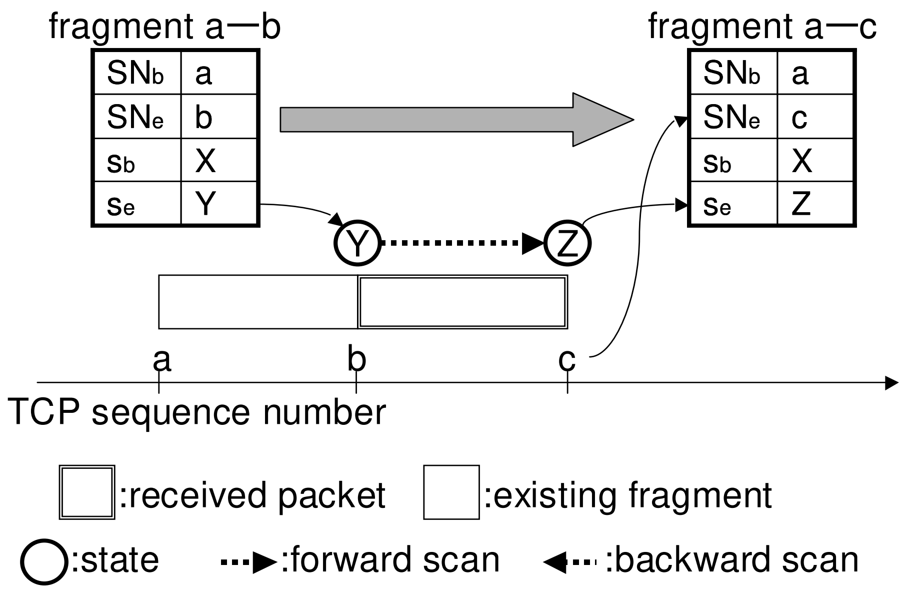

Figure 3: Packet processing for case PREC

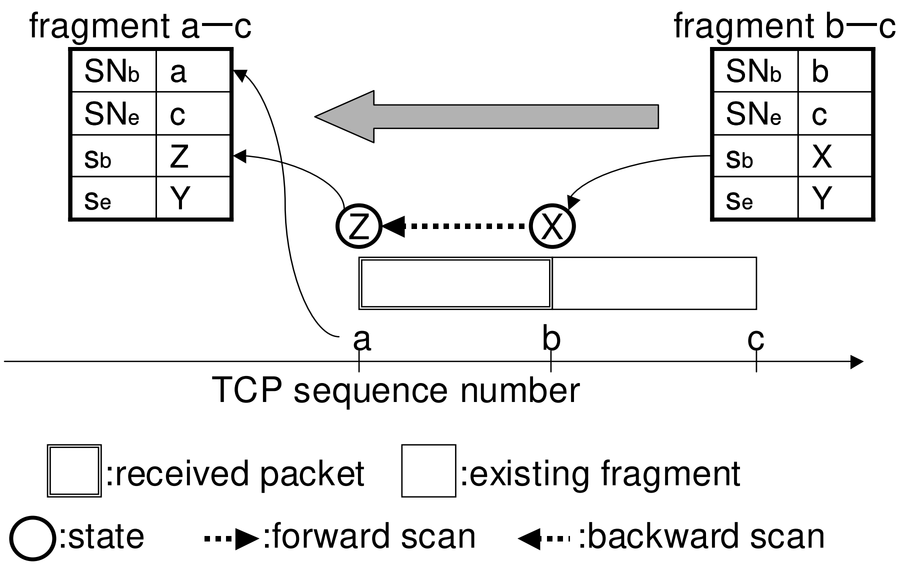

Figure 4: Packet processing for case SUCC

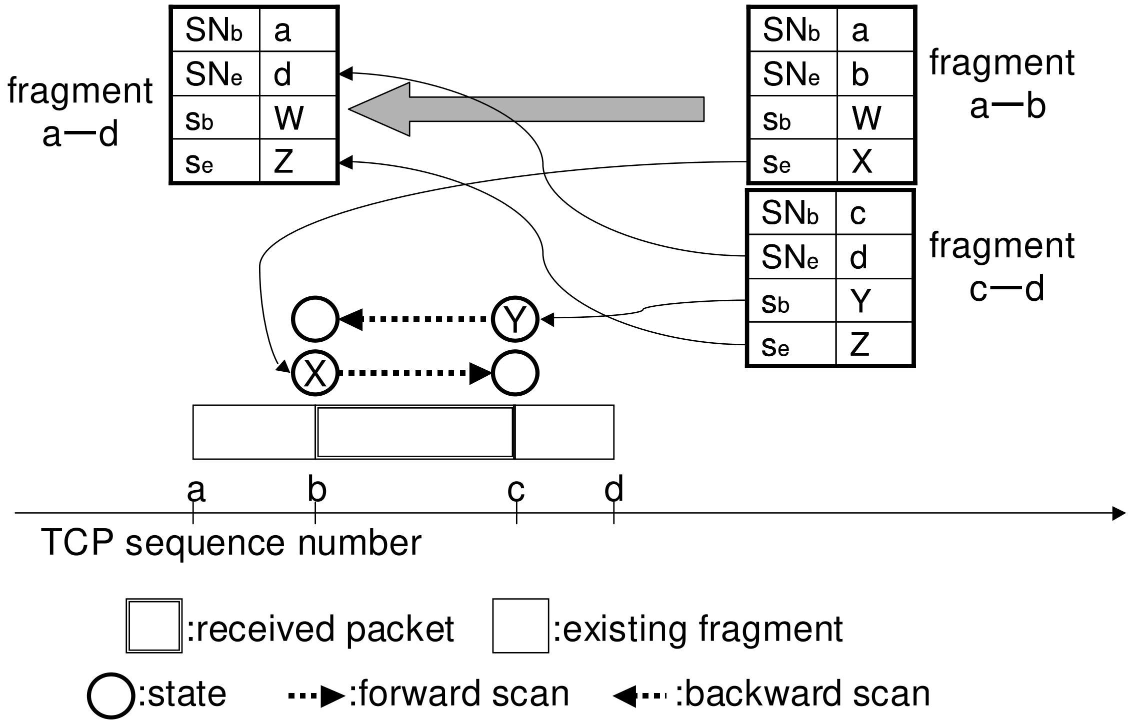

Figure 5: Packet processing for case CONNECT

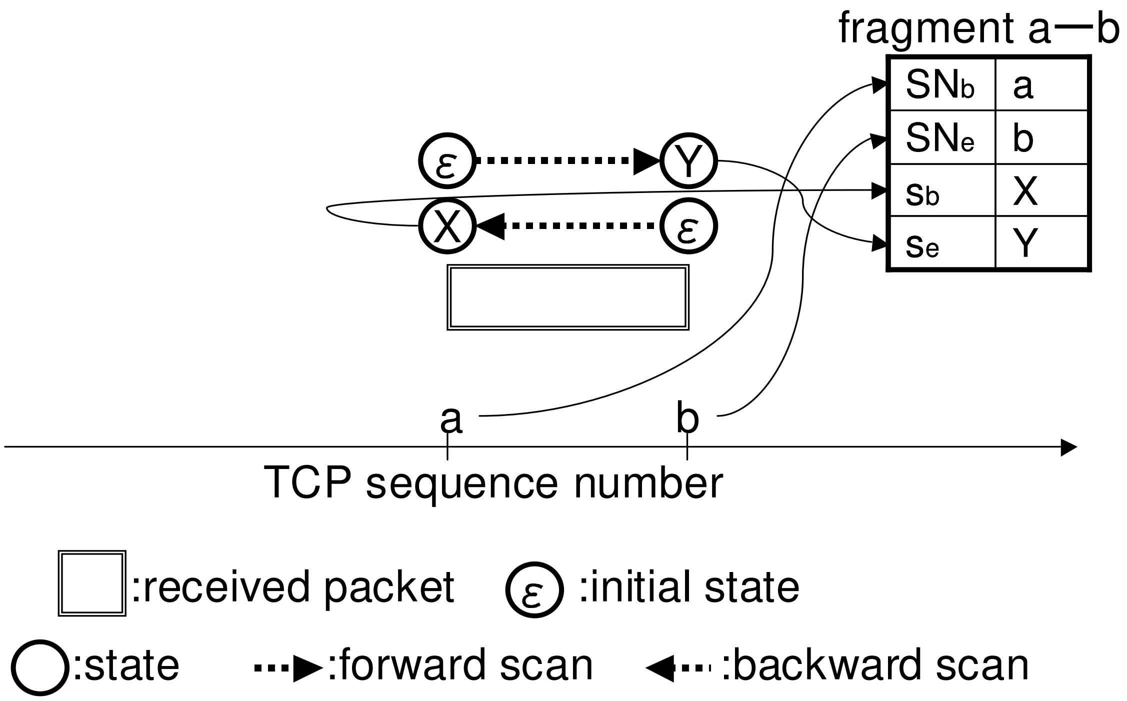

Figure 6: Packet processing for case NEW

3.3 Handling Small Packets

3.3.1 Characterizing the source of small packets

When the payload size of an arriving packet is smaller than $l-1$ octets, not all the target patterns can be discovered. More strictly, when there is one or more fragment that is shorter than $l-1$ octets, the overlook cannot be avoided without buffering all the octets of the small fragment.

The rule strings used in practical signature based NIDS are shorter than MTU of major networks. In an example of Snort [2], the maximum rule length is about 100 octets. Since the packet sizes of fast connections are normally set to MTU, they are longer than $l-1$. Therefore, the small packet problem is not experienced by high-speed connections. Just slow connections are involved in the problem.

3.3.2 Avoiding state-holding attack

To handle small packets, packet payload data octets must be saved. Therefore, it is a target of state-holding attacks. However, it is possible to guarantee the correctness of pattern match when additional memory is not available, at the cost of redundant retransmissions of TCP packets. Using this idea, following strategy is used.

- Available memory size for the small packet procedure is restricted to limit the damage of a state-holding attack to this procedure
- When a packet is arrived that requires buffering, and the available memory size is enough, packet octets are buffered.
- When a packet is arrived that requires buffering, and the available memory size is not enough, packet octets are discard.

The buffering is not necessary when packets are received in an increasing order of a sequence number. This stands even when there are small fragments whose octets are buffered. Therefore, innocent connections do not halted by the packet discard. In addition, though redundant packet retransmissions occur because of the packet discard, it does not congest the network because the procedure is not invoked by fast connections.

3.3.3 Procedure for each case

Received packets have to be processed in an appropriate way that depends on the packet size, and the sizes of neighbor fragments, if any. In the following, procedure of each case is described using a figure. In the figure, the packets are colored by its payload size, as shown in Table 3. In the following description, SMALL means that the number of octets is less than $l-1$, and LARGE means that the number is greater or equal to $l-1$. Though not described in the explanation, buffer memories are freed when they become unnecessary.

1. A new SMALL fragment is generated by a SMALL received packet. This case is shown in Figure 7. String matching cannot be performed yet. For the new fragment, $l-1$ octets buffer is allocated. The entire packet payload is saved in the buffer.
2. A SMALL fragment is produced by a SMALL received packet and one or more SMALL fragments. This case is shown in Figure 8. String matching cannot be performed yet. The entire octets of the new fragment is saved using a buffer of $l-1$ octets already allocated for either of the SMALL fragments.

color	description
white	The payload size is less than $l-1$
black	The payload size is greater than or equal to $l-1$
gray	The payload size is not concerned

Table 3: Meanings of packet colors

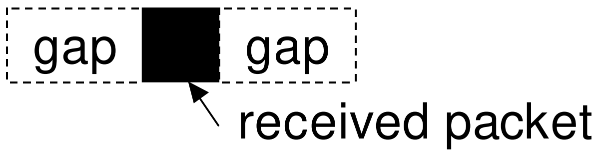

Figure 7: Fragment pattern of case 1.

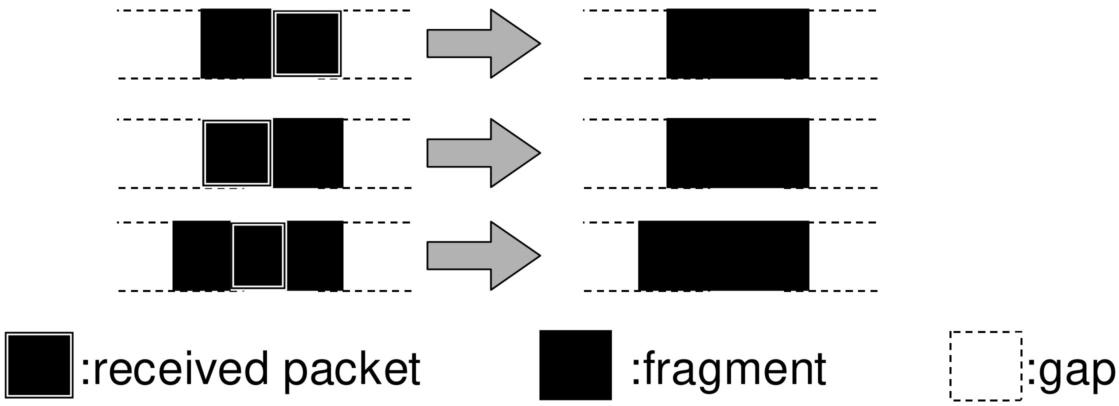

Figure 8: Fragment patterns of case 2.

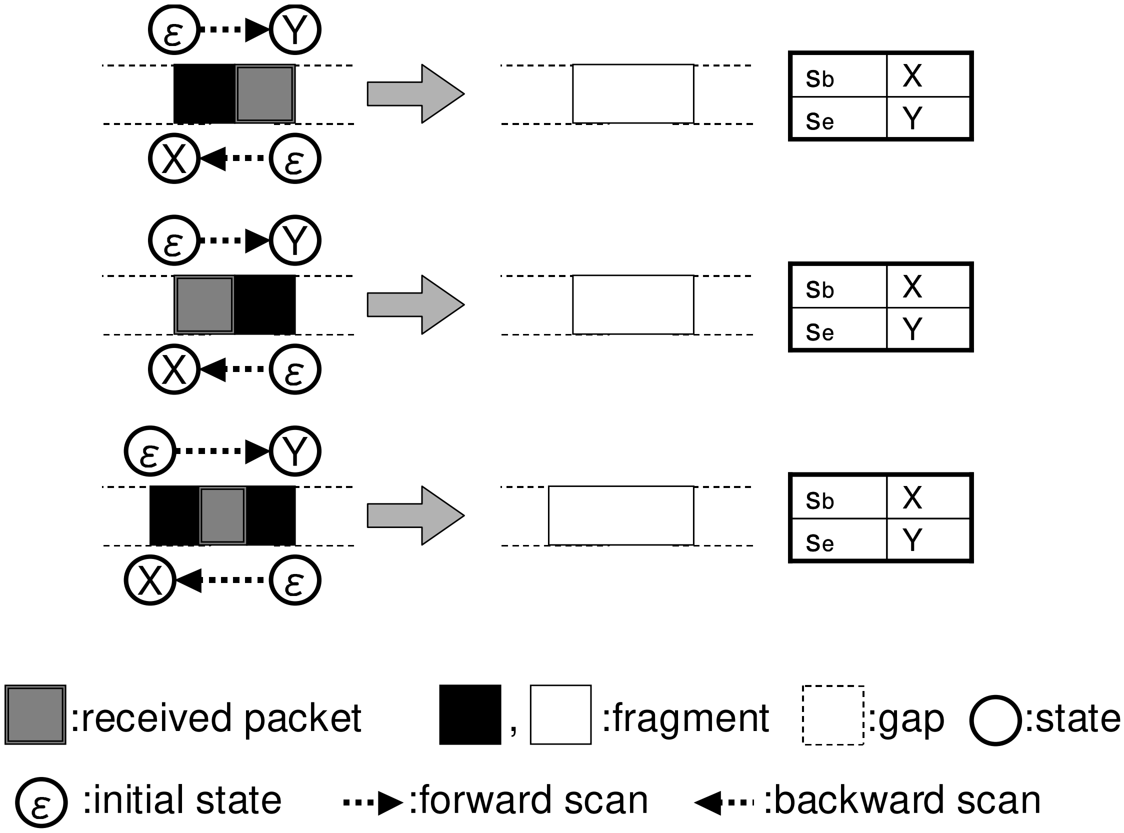

Figure 9: Fragment patterns of case 3.

3. A LARGE fragment is produced by a received packet and one or more SMALL fragments. This case is shown in Figure 9. Since the resulting fragment is LARGE, buffering is not necessary. Therefore, the new fragment information is calculated in the same way as the NEW case of the large packet procedure.

4. A LARGE fragment is produced by a received packet and one or more fragments. Exactly one of the fragments is LARGE. This case is shown in Figure 10. In this case, since the octets of the large fragment are not available, the search must be started from a boundary of it. Starting from the state at the boundary of the large fragment,

5. A LARGE fragment is produced by a SMALL received fragment and two LARGE fragments. This case is shown in Figure 11. In this case, to find target pattern that span the packet and both the fragments (such as p in Figure 11), special procedure is necessary. First, the forward matching unit calculates the ending state at the edge of the received packet (state Z in Figure 11). Since the state of SBT method indicates the longest substring matched up to the position, all the spanning patterns can be detected using this state and the ending state of the backward matching unit at this point (states Z and U in Figure 11).

Since the small packet procedure is not invoked by fast connections, the throughput is not required. Therefore, software implementation is enough. Using a software implementation, hardware resource usage can be eliminated.

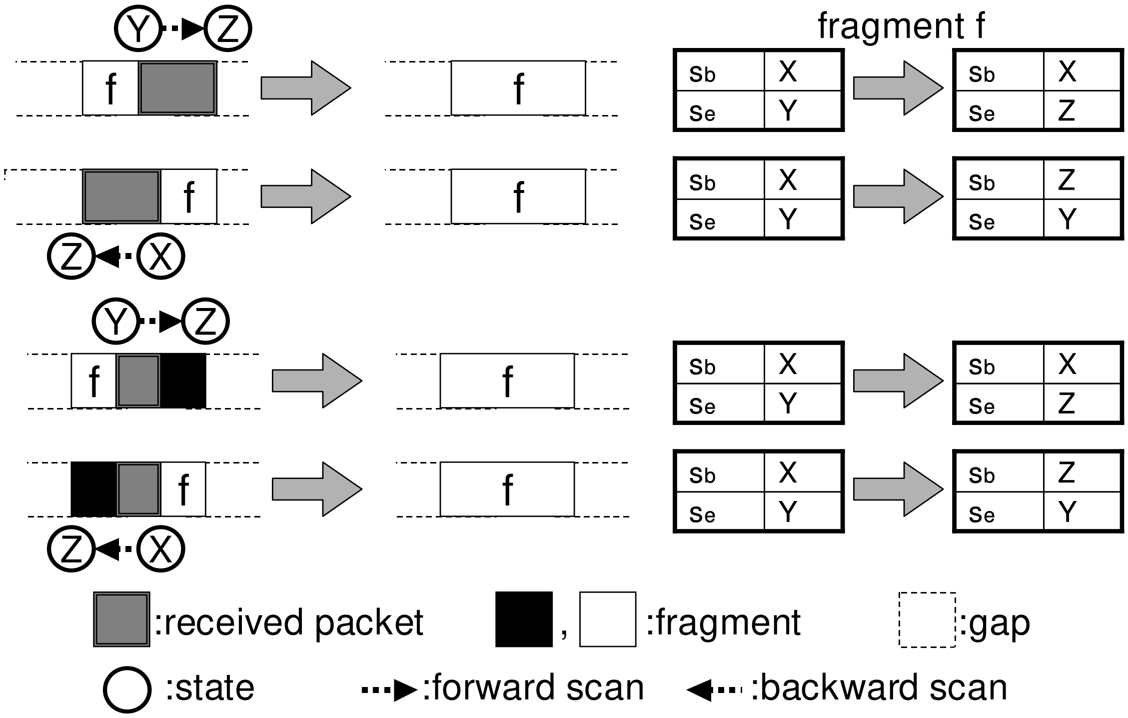

Figure 10: Fragment patterns of case 4.

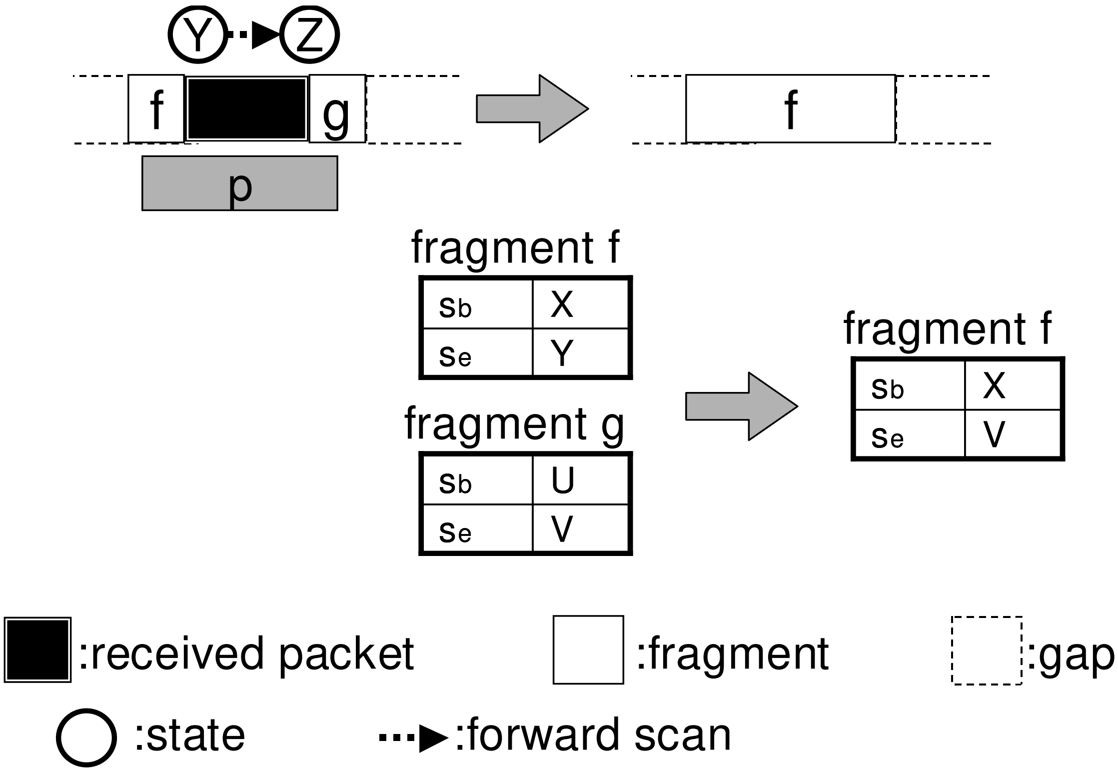

Figure 11: Fragment pattern of case 5.

4 Packet Signature

4.1 Virtually Strict Check using Hash Function

In TCP stream-level pattern match, retransmission consistency check is necessary to guarantee the correctness of the scan. For a strict check of the consistency, all the octets of first transmission are necessary. There is no way to always reduce the memory size.

However, *virtually* strict check is possible while reducing the memory size. That is, though the comparison procedure may return a wrong answer, the probability of the mistake can be limited to almost zero. By information theory, when s_s bits can be used to store an information of s_o bits data, the minimum theoretical mistake probability is $2^{-s_s} - 2^{-s_o}$. This minimum can be realized using random hash functions such as MD5 and DES. The strategy is as follows:

- The hash value of data in a first retransmission packet is calculated using a random hash function.
- When a retransmission packet is received, the hash value of its data is calculated using the same hash function. If the hash value is different from the first one, the payload data is different. If the hash value is same as the first one, the payload data is judged to be same as the first one, with the mistake probability of $2^{-s_s} - 2^{-s_o}$. s_s is the bit with of the hash value, and s_o is the number of payload bits of the packet.

We call the hash value of a packet as *signature*. Since attackers can intentionally force the matching unit to return a wrong answer when they know the hash function, it must be secret. In addition, it should be changed periodically to increase the security.

4.2 Handling Packet Boundary Mismatch

Since packet payload repacking is allowed in RFC793, the boundary of a retransmission packet may not match to those of first packets. In a normal connection, this seldom occurs [3]. In particular, it does not occur in a fast connection whose payload size is set to MTU. Nevertheless, since it is allowed by the standard, we must handle the exceptional cases.

In the following description, *signature block* means the octets to which a packet signature is calculated. Table 4 shows the data structure for each signature block [4]. When the boundary of a retransmission packet does not match to those of first packets, the packet signature cannot be applied. More strictly, the comparison is impossible when a subset of a signature block is contained in a retransmission packet. Figure 12 shows examples in which the comparison is possible, and Figure 13 shows ones in which the comparison is impossible.

When the comparison is impossible, the retransmission packet must not be passed to guarantee the security. Therefore, it must be buffered until it is checked. In addition, since the neighboring packet of the retransmission packet will be sent in future, the entire octets of the divided signature block must be saved for later use. As a result, the algorithm is constructed as follows:

1. When none of the edges of the received packet is at a middle of a signature block, the signature comparison is possible. The process is shown in Figure 14. First, the retransmission packet is separated at the boundaries of the existing signature blocks.

[3] Most of the case of repacking is that small packets are combined into a large one. In this case, the boundary of the large packet still matches to the first packets. Therefore, this case is not a problem.

[4] In our actual implementation, peep-hole optimizations are made to reduce memory usage. As a result, the actual data structure is slightly different from the table

SN_b	Sequence number of the starting octet of the block
SN_e	Sequence number plus 1 of the ending octet of the block
sig	Signature of the block
$octets$	Pointer to data structure for buffered octets, if any
$packets$	List of packets waiting for the octets of this block, if any

Table 4: Information stored for each signature block

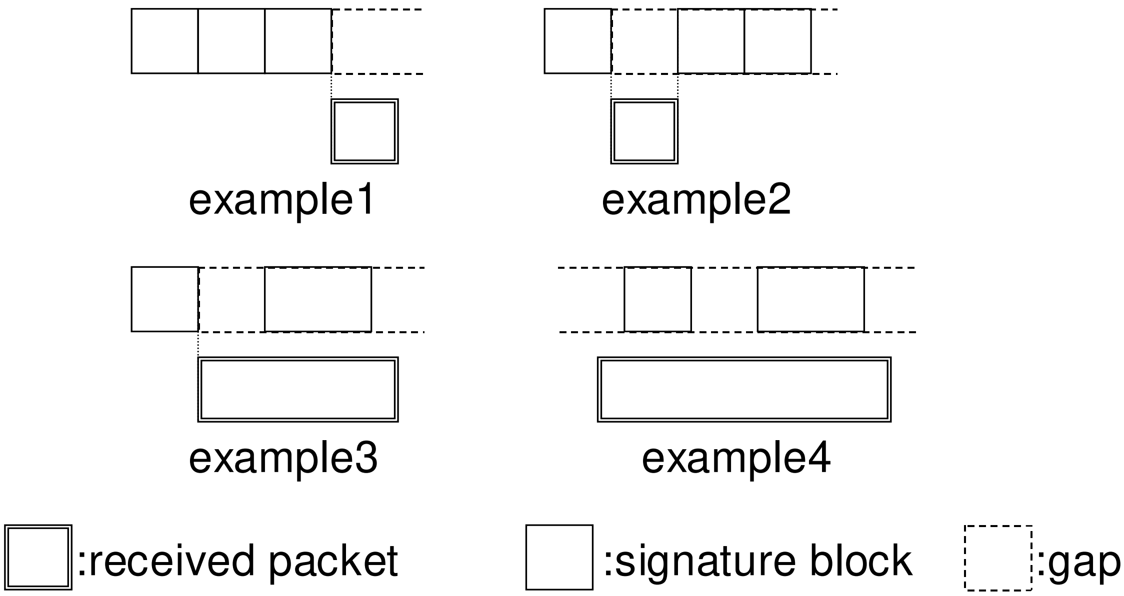

Figure 12: Examples in which packet signature is possible

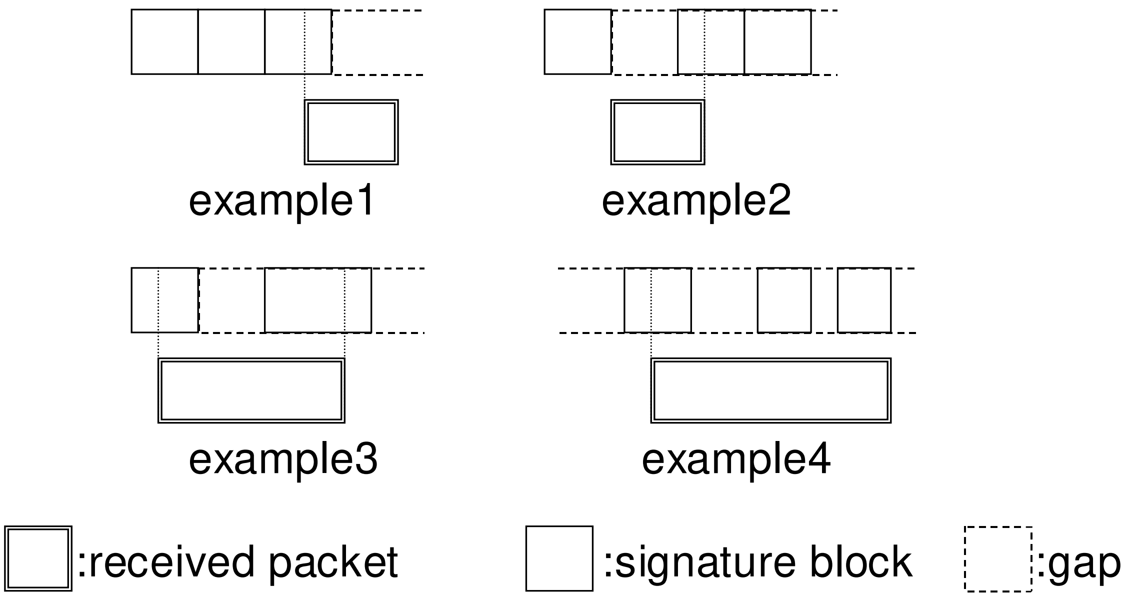

Figure 13: Examples in which packet signature is impossible

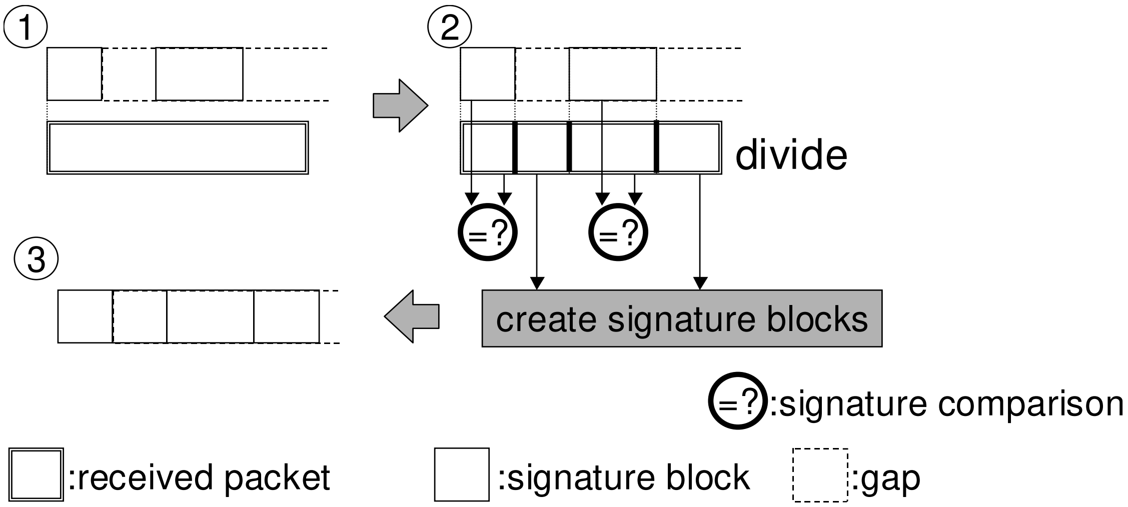

Figure 14: Packet reception procedure

For the received octets that are covered by a signature block, its signature is calculated and compared to the saved one. If the value is mismatched for either of the signature blocks, the packet is discarded or its data is corrected.

For the received octets that are not covered by a signature block, new signature blocks are created using the octets, and the signature of each block is calculated for later use.

2. When a edge of the received packet is at a middle of a signature block, the signature comparison is impossible just using the received octets at the edge. Therefore, this packet is buffered. In addition, the entire octets of the signature block are necessary. Thus it is marked using a flag. The other octets are processed in the same way as 1 before the packet is buffered.

3. When the retransmission octets covers a signature block that is marked in 2., The octets are added to the buffer. If the block is completely covered, the signature of all the pending packets are calculated and checked using the octets.

4.2.1 Avoiding state-holding attack

In the packet signature comparison procedure, following data is saved.

- Signature block information
- Buffered octets of signature blocks
- Buffered packets

When the memory allocation for the data is not restricted, it leads to a state-holding attack vulnerability. Therefore, when memory resource is insufficient, the memory allocation must be restricted. As a result, the procedure has to drop packets that cannot be checked due to the insufficient memory. We discuss the memory allocation control strategy for each data.

Signature Information When an existing signature block is discarded, the retransmission packets at the block must be discarded. As a result, the connection is aborted. Therefore, discarding existing signature blocks is difficult.

When existing signature blocks are not discarded, the problem is that new memory cannot be allocated for signature blocks of innocent connections. However, this problem is solved by allocating signature blocks for each connection beforehand. Since the number of the pre-allocated signature blocks limits the window size, the communication speed is degraded. However, the progress of the communication is guaranteed. Though the allocation strategy is to be discussed, in this research, a constant number of signature blocks are allocated for each connection for simplicity.

Buffered octets of signature blocks When existing signature block is discarded, a connection may be halted when (1)The boundary of a retransmission packet is different from those of first packets, *and* (2)Just a part of octets of a signature block has retransmitted. Since it is an exceptional race condition, innocent connections seldom halted by the discard. Therefore, we selected to discard old buffers to allocate a new buffer.

Buffered packets Buffered packets can be discarded without halting any connections because TCP retransmits the packet. Therefore, we selected to discard old buffered packets to buffer a new packet.

5 Evaluation

In this section, the proposed methods are evaluated in the throughput and the memory usage. The throughput evaluation is necessary to make sure that the methods are applicable to current high-speed networks. The memory usage evaluation is necessary to characterize the tolerance of the resulting system against state-holding attacks.

We implemented a prototype TCP stream scan system with bi-directional scanning and packet signature using XC2VP-50. SBT method [1] is used as the string match algorithm. We used Xilinx ISE6.3i tool suite for the synthesis and the post place and route performance analysis. Since the matching throughput depends on the input width of the SBT unit, we changed the input width in the evaluation.

5.1 Throughput

As we shown in 3, packets of fast streams are not processed in small packet handlers in bi-directional scanning method. Similarly, the packets of fast connections are not involved in the boundary mismatch handler of packet signature method. Therefore, these procedures are not implemented and not involved in the evaluation.

Figure 15 shows the performance result using a rule set of about 1000 characters. The packet signature width is 64 bits. Since fragment search and matching is not parallelized in the current implementation, the performance is reduced when packet loss and reorder happen. Nevertheless, this result shows that TCP stream scan on 10Gbps network can be realized using the proposed methods.

5.2 Tolerance against state-holding attacks

When the matching unit received a state-holding attack, it can be avoided that innocent connections are aborted, as long as the number of connections is less than a threshold. However, the window sizes of the connections are limited due to the insufficient memory resource. As a result, the throughput of each connection is decreased. Therefore, the window size reduction must be minimized.

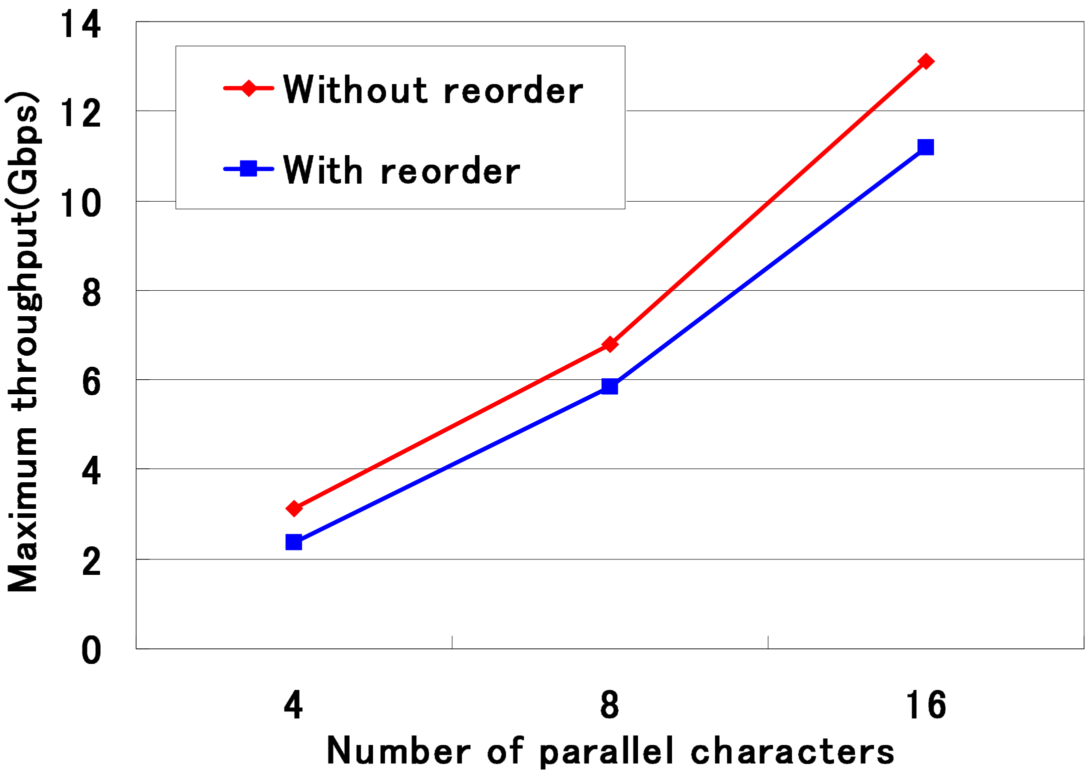

Figure 15: Maximum throughput of the TCP stream scan unit

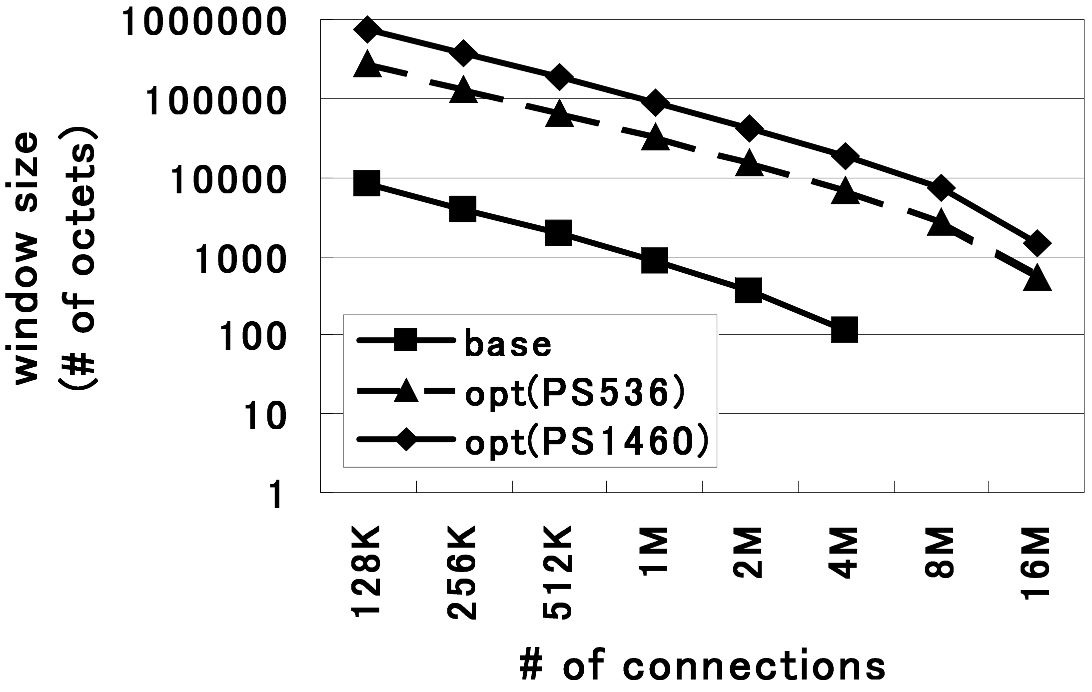

Figure 16: Number of connections and guaranteed window size

Figure 16 shows the relation between the number of connections and minimum guaranteed window size per connection. Total available memory size is 1GB. Other conditions are same as those used in Figure 15. In the figure, "base" is a configuration without memory size optimization, and "opt" is a system based on SBT, bi-directional scanning, and packet signature. The numbers after "PS" are the payload size. The payload size of 536 octets is evaluated because this is the minimum guaranteed MTU of TCP/IP, and the worst case for the "opt" configuration. The payload size of 1460 octets is a value of the Ethernet, which is a widely used standard.

In the figure, the opt(PS536) always provides a window size that is more than 30 times greater than the non-optimized case. In the case of opt(1460), the difference is about 100 times. Therefore, using the proposed method, higher communication throughput is provided compared with a non-optimized case under a given number of streams and a memory size. Therefore, using our method, high communication throughput is realized under a state-holding attack.

A problem to be solved is connection-flooding attacks. When the number of connections exceeds a threshold, the available window size drops to zero, i.e. communication is impossible. In the base configuration, the threshold is 1.2M connections, and in both the optimized configurations, it is 7.7M connections. Though the threshold value is improved by our method, it is meaningless once attackers inflate the number of connections.

When the threshold is reached, the matching unit must halt part of existing connections to get a free area. As a result, parts of innocent connections are halted. To resolve the problem, it is necessary to judge whether each connection is "innocent" or not. This is our future work.

6 Related Works

Necker et al. [14] proposed a TCP stream reassembling module using hardware. However, separated instance of the reassemble unit is necessary for each connection. As a result, the number of connections is limited. In addition, retransmission inconsistency check is not performed. Furthermore, since FPGA on-chip memory is used for reassemble buffer, the available window size is limited.

TCP-Splitter [16] is an FPGA based TCP reassembling module. To handle lost or reordered packets without using reassemble buffer, they discard the packets that are not received in-order. As a result, many redundant retransmissions will be triggered. In addition, retransmission inconsistency check is not performed. On the other hand, using our method, when there is no DoS attack, redundant packet discard is not performed. In addition, retransmission inconsistency check is performed using small amount of memory.

Though we used normalization [11] [17] technique to remove ambiguity of inconsistent retransmission, our approach is also applicable to Active-mapping [18].

7 Concluding Remarks

In this paper, we proposed techniques to realize TCP stream level pattern match for signature based NIDS. The techniques are bi-directional scanning and packet signature. Different from the existing methods, the proposed methods enable both high throughput and high tolerance against state-holding attacks. These features are essential to enable TCP stream level pattern match in signature based NIDS.

Bi-directional scanning is the method to handle TCP packet loss and reorder while eliminating buffering requirement. Packet signature is a method to make a virtually strict check of retransmission consistency while reducing the necessary memory size.

A complete TCP stream pattern match unit is implemented using the proposed methods. In the throughput evaluation, using XC2VP-50 FPGA, the matching unit achieved 13.1Gbps when packets are received in-order, and 11.2Gbps when packets are reordered. This result indicates that the proposed methods are applicable to recent 10Gbps-class high-speed networks. In the evaluation of tolerance against state-holding attacks, we found that the proposed system provides 30–100 times higher communication throughput under a state-holding attack compared with a system without the proposed methods. This result shows a good tolerance against state-holding attacks.

7.1 Future Work

A problem to be solved is connection-flooding attack. When the number of connections reaches a particular threshold, part of connections must be halted, including innocent ones. Though the threshold connection number is increased by 6.4 times using the proposed methods, it does not guarantee that the threshold is not reached. We will work around for this problem as our future work. In addition, we are currently implementing complete network equipment including the TCP stream level pattern match unit using an FPGA-based network card.

Acknowledgements

This research is partially supported by the Special Coordination Fund for Promoting Science and Technology, CREST project of Japan Science and Technology Corporation, and 21st century COE project of Japan Society for the Promotion of Science.

References

[1] Y. Sugawara, M. Inaba, K. Hiraki, "Over 10gbps string matching mechanism for multi-stream packet scanning systems," in *Proc. of 14th Intl. Conf. on Field Programmable Logic and Applica tions(FPL '04)*, August 2004.

[2] M. Roesch, "Snort - Lightweight Intrusion Detection for Networks," in *Proc. of Lisa'99: 13th Administration Conference*, 1999.

[3] Julia Allen, Alan Christie, William Fithen, John McHugh, Jed Pickel, Ed Stoner, "State of the Practice of Intrusion Detection Technologies," tech. rep., Software Engineering Institute, Carnegie Mellon University, January 2000. http://www.sei.cmu.edu/publications/documents/99.reports/99tr028/99tr028abstract.html.

[4] M. Bykova, S. Ostermann, Brett Tjaden, "Detecting network intrusions via a statistical analysis of network packet characteristics," in *Proc. of 33rd Southern Sysmposium on System Theory (SSST 2001)*, March 2001.

[5] "Dragon NIDS." http://www.enterasys.com/products/ids/whitepapers/.

[6] "Hogwash NIDS." http://sourceforge.net/projects/hogwash/.

[7] Sebastian Elbaum and John, C. Munson, "Instrusion detection through dynamic software measurement," in *Proceedings of the Workshop on Intrusion Detection and Network Mintoring*, April 1999.

[8] David Marchette, "A statistical method for profiling network traffic," in *Proceedings of the Workshop on Intrusion Detection and Network Mi ntoring*, April 1999.

[9] J. Postel, "RFC791 : Internet Protocol," September 1981.

[10] "Fragrouter download page in SecurityFocus." http://www.securityfocus.com/tools/176.

[11] M. Handley, V. Paxson, "Network Intrusion Detection : Evasion, Traffic Normalization, and End-to-End Protocol Semantics," in *Proc. of 10th USENIX Security Symposium*, August 2001.

[12] Thomas H. Ptacek and Timothy N. Newsham, "Insertion, evasion, and denial of service: Eluding network intrusion detection," January 1998. http://www.insecure.org/stf/secnet ids/secnet ids.html.

[13] D. V. Schuehler, J. Moscola, J. Lockwood, "Architecture for a Hardware Based, TCP/IP Content Scanning System," in *Proc. of 11th IEEE Symp. on High Performance Interconnects (HotI '03)*, pp. 89 – 94, August 2003.

[14] M. Necker, D. Contis, D. Schimmel, "Tcp-stream reassembly and state tracking in hardware," in *Proc. of 10 th Annual IEEE Symp. on Field-Programmable Custom Computing Machines (FCCM'02)*, pp. 286–287, September 2002.

[15] J. Postel, "RFC793 : Transmission Control Protocol," September 1981.

[16] D. V. Schuehler, J. Lockwood, "TCP-Splitter: A TCP Flow Monitor in Reconfigurable Hardware," in *Proc. of High Performance Interconnects (HotI '02)*, pp. 127–131, August 2002.

[17] G. R. Malan, D. Watson, F. Jahanian, and P. Howell, "Transport and Application Protocol Scrubbing," in *Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2000)*, vol. 3, pp. 1381–1390, March 2000.

[18] Umesh Shankar, Vern Paxson, "Active Mapping: Resisting NIDS Evasion Without Altering Traffic," in *Proc. of the 2003 IEEE Symposium on Security and Privacy*, May 2003.