Secure SCTP against DoS Attacks in Wireless Internet *

Inwhee Joe

College of Information and Communications
Hanyang University
Seoul, Korea
iwjoe@hanyang.ac.kr

Abstract. The Stream Control Transport Protocol (SCTP) is a new transport layer protocol that has been designed to provide reliable transport over the Internet. While the Transport Control Protocol (TCP) is the most popular transport protocol for the Internet, it falls short with regard to security, more specifically resilience to Denial-of-Service (DoS) attacks, such as SYN attacks. The need for resilience to DoS attacks is obvious, and SCTP provides for this resilience via its improved handshake mechanism and the Cookie feature. This paper discusses the SCTP simulation with particular emphasis on resilience to DoS attacks. As revealed by our detailed simulation study, the increased DoS resilience comes with increased overheads. While DoS resilience is extremely critical, reducing overheads in the resource-constrained wireless environment also assumes paramount importance. Hence we propose secure SCTP with an innovative Cookie mechanism using a combination of cache and INIT packet repetition to minimize the communication overhead and simultaneously to maximize security associated with SCTP's DoS resilience for wireless Internet.

1 INTRODUCTION

The Stream Control Transport Protocol (SCTP) [7] is a new transport-layer protocol that is being designed to provide reliable and secure transport of a variety of applications over IP networks. While the Transport Control Protocol (TCP) as its counterpart is the most popular and widely used transport protocol in the IP networks [3], SCTP not only provides the features offered by TCP but also has additional important characteristics. Examples include SCTP's novel features such as multi-homing and multi-streaming, built-in "protocol-hooks" to provide resistance to DoS (Denial of Service) attacks as well as the capability of supporting various ordering types (i.e., strict ordering, partial ordering and un-ordered delivery types).

In this paper, we use simulation techniques to investigate and analyze the SCTP's DoS feature in the wireless Internet. To this end, we have constructed

* This work was supported by grant No. IITA-2005-C1090-0501-0022 from the ITRC Support Program of the Ministry of Information and Communication.

Please use the following format when citing this chapter:

Joe, I., 2006, in IFIP International Federation for Information Processing, Volume 213, Network Control and Engineering for QoS, Security, and Mobility, V, ed. Gaïti, D., (Boston: Springer), pp. 65–74.

detailed simulation models of SCTP to generate DoS-attack scenarios in a wireless Internet model consisting of wireless workstations connected by a wireless LAN as a preliminary simulation study. Our detailed simulation analysis has helped reveal the overheads/bottlenecks associated with SCTP's DoS scenarios and to arrive at an innovative Cookie mechanism (via a combination of cache and INIT packet repetition) to minimize the overhead and to maximize security at the same time.

The remainder of this paper is organized as follows. Section 2 presents the SCTP simulation and resulting scenarios developed using OPNET as well as overhead and delay-analysis. Section 3 proposes secure SCTP with an improved Cookie mechanism (based on an innovative combination of cache and INIT packet repetition) to help reduce the DoS related overheads and thereby render SCTP an extremely viable solution in the resource-constrained wireless Internet environment. Section 4 concludes the paper.

2 SCTP SIMULATION

First, we will compare TCP and SCTP briefly in terms of connection estalishment. In TCP, the 3-way handshake sequence is used to set up TCP connections. To open a connection, the TCP client initiates a connection establishment procedure as an active opener by sending a SYN packet to the TCP server. The SYN packet carries connection initialization information like the initial value of the sequence number. Then, the TCP server waits for an acknowledgment (ACK) from the TCP server, indicating that the SYN packet has been received. After receiving the SYN packet, the TCP server sends its SYN packet along with the acknowledgment, indicating that it is also ready to accept the connection as a passive opener. Finally, the TCP client sends its ACK packet in response with the SYN/ACK packet from the TCP server. Once this exchange is complete, the connection is fully established and data can be transferred through this connection.

In case of TCP, whenever a connection establishment request arrives, the TCP server just allocates for each connection all the resources including memory for the transmission control block (TCB), before it verifies the TCP client. Assume that there is an intruder, trying to attack the TCP server by sending a bunch of SYN packets. Since TCP implementations limit the number of connections due to the resource problem, it could cause the server to use up memory and resources handling new connection requests. As a result, the TCP server reaches its limit so that it cannot accept any new incoming connections, leading to the state of the denial of service (DoS).

On the other hand, SCTP relies on the 4-way handshake sequence instead of the 3-way handshake of TCP, where a cookie mechanism is incorporated into the sequence to guard against some types of DoS attacks. To start an association (or connection in the TCP terminology), the SCTP client initiates the 4-way handshake by sending an INIT packet to the SCTP server. The INIT packet carries association initialization information including the initial sequence number and

receiver window. Likewise, the SCTP client waits for an acknowledgment (ACK) from the SCTP server, indicating that the INIT packet has been received. After receiving the INIT packet, the SCTP server sends its INIT-ACK packet along with the Cookie as a variable parameter in it. The Cookie parameter contains the minimal TCB information required to create the association and a message authentication code (MAC). The MAC code is generated using a hash algorithm (e.g., MD5 or SHA-1 algorithms) with the input of the TCB information and a secret key.

When the SCTP client receives the INIT-ACK packet from the SCTP server, it puts the servers Cookie into a COOKIE-ECHO packet as it is and returns to the server. At the same time, if there is any data to send, it can be included in this packet for transmission efficiency. Upon reception of the COOKIE-ECHO packet, the SCTP server can validate the Cookie by checking the MAC code and uses it to rebuild the TCB. Since the SCTP client has been verified through the cookie mechanism at this point, the server allocates all the resources and memory right away. Then, the server sends a COOKIE-ACK packet to the SCTP client (optionally bundling any data with this packet for transmission efficiency). Once this last exchange is done, the association is fully established and data can be transferred through this association.

This section presents SCTP simulation using OPNET to demonstrate how SCTP performs in response to DoS attacks and also in normal situation compared to TCP, focusing on the connection establishment phase. As shown in Fig. 1, our network model is built on one wireless LAN (IEEE 802.11) as our preliminary study for a wireless Internet model, which contains 20 wireless workstations and one of them is an intruder. The Access Point (AP) in the wireless LAN is connected to the IP router and the File Server on the 100Base-T wireline LAN operating at 100 Mbps. The wireless workstations in the wireless LAN communicate with the File Server through the AP and the IP router.

To compare the security feature between TCP and SCTP in the scenario of DoS attacks, we develop two types of node models for the wireless workstation. The two node models differ in the transport protocol, namely one employs TCP and the other SCTP. The node model represents a wireless workstation with file transfer applications running as a client over TCP/IP or SCTP/IP according to the type of the node model. The wireless workstation supports one underlying wireless LAN connection at 1 Mbps. Likewise, we also develop two types of node models for the File Server, which represents a server node with file server applications running over TCP/IP or SCTP/IP.

The scenario is that the Intruder in the wireless LAN attacks the File Server by sending a large number of association or connection establishment requests (i.e., INIT packets for SCTP and SYN packets for TCP, respectively) with the forged IP source address, until the File Server has reached its limit on the number of connections (due to resource problems). In case of TCP, since the TCP server (File Server) allocates all the resources right away before it verifies the TCP client, it causes the server to very rapidly become unable to accept any new incoming connections, declaring itself as the DoS state. Even if this situation

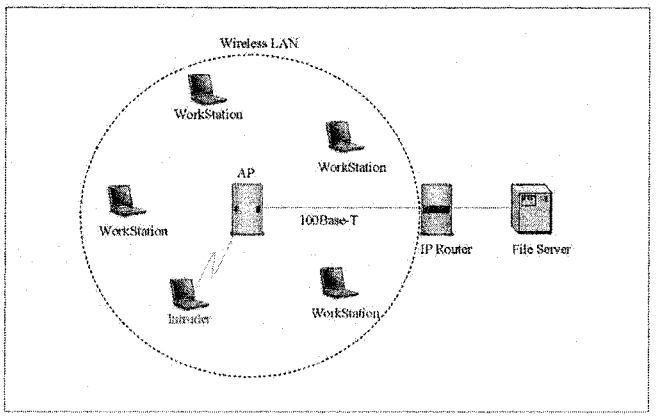

Fig. 1. Simulation Network Model

is cleared out after a period of time, the Intruder can send SYN packets fast enough so that it ensures to make the situation recur. The time period can be obtained using the current values of the TCP simulation parameters in OPNET, as shown in Fig. 2.

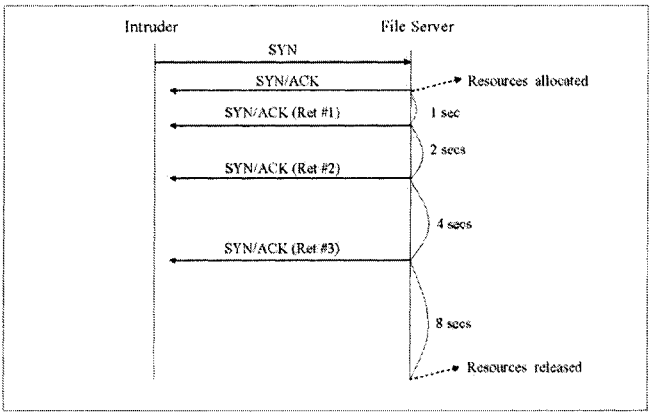

Fig. 2. Time Sequence Diagram for SYN Retransmission in TCP

When a SYN packet arrives from the Intruder, the File Server allocates all the resources right away and then sends to the Intruder its SYN packet with the acknowledgment, indicating that the SYN packet has been received successfully.

At the same time, the File Server starts a retransmission (RTO) timer whose value is set initially to 1 second in the TCP parameters with OPNET simulation. Since the Intruder uses a forged source IP address, the SYN/ACK packet from the File Server gets nowhere and therefore, there is no ACK for this packet. As a result, the retransmission timer on the File Server expires, and then it triggers a retransmission of the SYN/ACK packet to the Intruder using the same forged source IP address. At this time, the value of the retransmission timer is doubled to 2 seconds based on the TCP protocol.

Since the maximum number of retransmission attempts on connection establishment is restricted to 3 in the TCP parameters, the File Server gives up after 3 times of retransmission. At this point, it releases all the resources allocated before and reports an error to the upper layer. The value of the retransmission timer is doubled every time it expires, because there is no ACK packet received from the other side (Intruder). In summary, with the current values of the TCP simulation parameters in OPNET, it takes 15 seconds to clear out the whole situation once it starts from the time when the resources are allocated first. Also, there is a limit on the number of connections, which is 200 as a configurable system parameter. Therefore, if the Intruder sends more than 200 SYN packets within 15 seconds, then it causes the File Server to reach the limit, rendering it unable to accept any new incoming connections as the DoS state.

In the OPNET simulation, the Intruder is designed to generate SYN packets at the rate of 15 packets/s, which is faster than the rate of 200/15 packets/s above. The simulation results show that the File Server has entered the DoS state after it receives 200 SYN packets from the Intruder. On the other hand, since SCTP unlike its counterpart TCP allocates the resources only after the SCTP client is confirmed with the Cookie mechanism, the Intruder is prevented from "hogging" up system resources. Our simulation studies create the above scenario for the SCTP case and we validate via the OPNET-based simulation models developed in this work that SCTP indeed avoids this situation. Furthermore, unlike the TCP case, the SCTP system continues to accept new and valid incoming association requests. Finally, after a certain maximum number of retransmission attempts, the File Server gives up and reports an error to the upper layer. No resource-release phase is entered now since there were no resources allocated to begin with (to the intruding end station).

In fact, more increased security comes at the cost of more increased overhead in the transport layer, as shown in Table 1. In terms of the number of packets, TCP and SCTP have the same overhead based on the protocol behavior. However, if the packet size is considered in the overhead comparison, SCTP causes more overhead than TCP. To make the equal conditions, it is assumed that only mandatory parameters are used for both cases. Since there is no data carried in this connection setup phase, the TCP packet contains only the packet header of 20 bytes. In SCTP, the packet size is different according to the packet type. The INIT packet is 32 bytes and the INIT-ACK packet is 32 bytes plus a variable Cookie parameter, which is up to 156 bytes depending on the implementation. Since the bandwidth is a scarce resource in the wireless environment, the Cookie

parameter should be minimized. In the next section, we present a novel efficient and secure Cookie mechanism for wireless Internet.

Table 1. Overhead Table for TCP vs. SCTP in DoS'Attack

	TCP SYN	SCTP INIT	TCP SYN/ACK	SCTP INIT-ACK
Number of Packets	1	1	4	4
Packet Size	20 bytes	32 bytes	20 bytes	32 bytes + Cookie

Even in normal situation, SCTP causes more overhead at the connection setup phase because of the 4-way handshake and Cookie mechanisms. To establish a connection, SCTP consumes 4 packets instead of 3 packets of TCP in terms of the number of packets. For the packet size, the total overhead is 60 bytes in case of TCP, because each TCP packet is 20 bytes and the 3-way handshake sequence is used. On the other hand, the total overhead in SCTP comes to 96 bytes plus twice the Cookie parameter, because the 4-way handshake sequence is used instead with the INIT packet of 32 bytes, the INIT-ACK packet of 32 bytes plus Cookie, the COOKIE-ECHO packet of 16 bytes plus Cookie, and the COOKIE-ACK packet of 16 bytes. Therefore, SCTP has 36 bytes plus twice the Cookie more overhead than TCP in terms of the packet size.

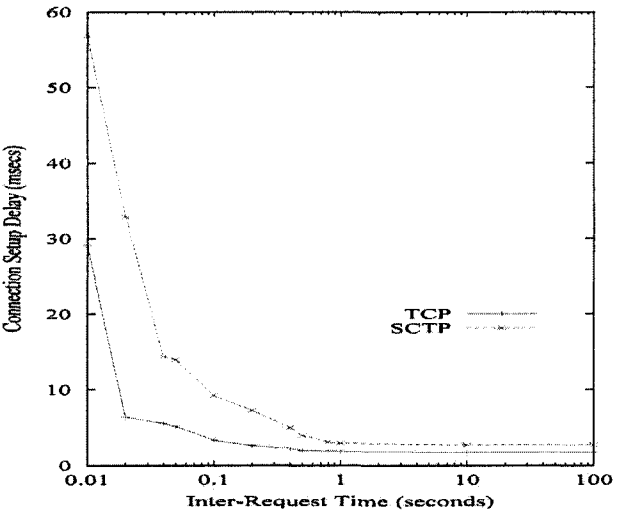

Fig. 3. Connection Setup Delay for SCTP vs. TCP in Normal Situation

We also measure connection setup delays for SCTP versus TCP as a function of offered load in normal situation over the simulated wireless network. The

connection setup delay denotes the entire duration measured from the time a connection setup request packet is sent to the time the transport connection is established successfully. The offered load is defined as data traffic offered to the transport layer by the file transfer application on the client side by changing the mean value of the IRT (Inter-Request Time) according to the exponential distribution. In SCTP, it is assumed that the Cookie parameter is fixed as 32 bytes. In Fig. 3, the simulation results exhibit the exponentially increasing delay behavior on connection setup with the higher load for both SCTP and TCP, where the shorter IRT interval means the higher load. Since SCTP uses the 4-way handshake sequence, it causes more queuing delay at the higher load, compared to the 3-way handshake TCP. Under light load conditions, SCTP takes about 1 msec more delay than TCP and the simulation results approach the theoretical minimum delay values, which can be obtained by computing transmission times only (packet size/bit rate). Those minimum values are 1.5 msecs for TCP and 2.56 msecs for SCTP, respectively.

3 SECURE SCTP FOR WIRELESS INTERNET

Here, we propose secure SCTP with an efficient and secure Cookie mechanism using an innovative combination of cache and INIT packet repetition to minimize the communication overhead and simultaneously to maximize security for wireless Internet. In essence, since the transmission control block (TCB) information consumes a large bandwidth and contains sensitive connection information, the Cookie parameter is modified to exclude any TCB information inside for transmission efficiency and security purposes. Instead, the INIT packet is repeated to recreate the TCB information later just in case of the DOS attacks, and also the TCB cache is used to significantly reduce the connection set-up delays in normal situation. First, we will describe briefly how TCP and SCTP establish connections, before we get into the details of our proposed Cookie mechanism.

In TCP, the 3-way handshake sequence is used to set up TCP connections. To open a connection, the TCP client initiates a connection establishment procedure as an active opener by sending a SYN packet to the TCP server. The SYN packet carries connection initialization information like the initial value of the sequence number. Then, the TCP client waits for an acknowledgment (ACK) from the TCP server, indicating that the SYN packet has been received. After receiving the SYN packet, the TCP server sends its SYN packet along with the acknowledgment, indicating that it is also ready to accept the connection as a passive opener. Finally, the TCP client sends its ACK packet in response with the SYN/ACK packet from the TCP server. Once this exchange is complete, the connection is fully established and data can be transferred through this connection.

In case of TCP, whenever a connection establishment request arrives in the form of SYN packet, the TCP server just allocates for each connection all the resources including memory for the TCB without verifying the TCP client. Assume that there is an intruder, trying to attack the TCP server by sending a

bunch of SYN packets (so called SYN attack). Since TCP implementations limit the number of connections due to the resource problem, eventually it could cause the server to use up all the resources. As a result, the TCP server cannot accept any new incoming connections, leading to the state of the denial of service (DoS).

On the other hand, SCTP relies on the 4-way handshake sequence instead of the 3-way handshake of TCP, where a "Cookie" mechanism is incorporated into the sequence to guard against some types of DoS attacks. To start an association (or connection in the TCP terminology), the SCTP client initiates the 4-way handshake by sending an INIT packet to the SCTP server. The INIT packet carries association initialization information including the initial sequence number and receiver window. Likewise, the SCTP client waits for an acknowledgment (ACK) from the SCTP server, indicating that the INIT packet has been received. After receiving the INIT packet, the SCTP server responds with its INIT-ACK packet along with the Cookie as a variable parameter in it. The Cookie parameter contains the minimal TCB information required to create the association and a message authentication code (MAC). The MAC code is generated using a hash algorithm (e.g., MD5 or SHA-1 algorithms) with the input of the minimal TCB information and a secret key [4].

When the SCTP client receives the INIT-ACK packet from the SCTP server, it puts the received Cookie into a COOKIE-ECHO packet as it is and sends the packet to the server. At the same time, if there is any data to send, it can be included in this packet for transmission efficiency. Upon reception of the COOKIE-ECHO packet, the SCTP server first validates the Cookie by the MAC code and its lifespan, and then uses it to rebuild the TCB information only if the Cookie is valid. Since the SCTP client has been verified through the Cookie mechanism at this point, the server can allocate all the resources and memory right away. Then, the server sends a COOKIE-ACK packet to the SCTP client (optionally bundling any data with this packet for transmission efficiency). Once this last exchange is done, the association is fully established and data can be transferred through this association.

The Cookie mechanism is employed to guard specifically against the DoS attacks flooding with INIT packets. Rather than allocating resources for the INIT packet, the server instead creates a Cookie parameter with the TCB subset in it and sends it back in the INIT-ACK packet. Since this packet goes back to the source address of the INIT packet, the Intruder with a forged IP address will not get the Cookie. Only a valid SCTP client will get the Cookie and return it in the COOKIE-ECHO packet. In this case, the SCTP server checks its integrity by the MAC code and its validity by the lifespan. If it is good, the server uses it to rebuild the TCB for this connection by allocating all the resources at this time, avoiding the DoS attacks as a result.

The Cookie parameter is defined in a Type-Length-Value format, because it is variable. The first two fields of Type and Length take 4 bytes, and the Value field of Cookie consists of 4 parts: TCB subset, MAC, Timestamp, and Life span. The TCB subset represents the minimal subset of TCB information necessary

to recreate the TCB. Its size is variable up to 140 bytes, which is the entire TCB data. The remaining three parts of the Cookie parameter are fixed: 4 bytes of MAC, 4 bytes of Timestamp, and 4 bytes of Lifespan as typical size values. The Cookie creation timestamp and the lifespan are used together to check whether the received Cookie is stale or not, while the MAC code is used to check the Cookie integrity.

According to the 4-way handshake sequence of SCTP, the Cookie parameter is echoed between client and server in the INIT-ACK and COOKIE-ECHO packets. Since the Cookie parameter includes the large TCB data up to 140 bytes, such round trip causes a large overhead especially for wireless networks, where the bandwidth is a scarce resource. The proposed Cookie mechanism is to exclude the TCB data from the Cookie parameter to substantially reduce the communication overhead. At the same time, since the TCB information carries sensitive connection information like source identity and initial sequence number, the best way for security is to get rid of it completely so that this information may not be exposed at all to outside. Another point of our mechanism is to use a TCB cache in order to speed up the connection setup operation. That is, if there is a cache hit, there is no need to recreate the TCB in this case. Most of time, cache hits are expected in normal situation except for the DoS attacks or near-capacity situation. The TCB cache is implemented in S/W just like the buffer cache concept and the total number of TCB cache is the same as the limit on the number of connections.

To begin with, the TCB cache is initialized as an empty list. If the INIT packet is received, the SCTP server allocates a cache entry to create a new TCB based on the connection information from this packet. The source IP address and port can be used as an index key to the TCB cache in the cache pool. In response, the INIT-ACK packet is sent along with the Cookie parameter in it without the variable TCB part, as mentioned above. After that, the client sends the COOKIE-ECHO packet with the Cookie parameter as it is and the repeated INIT packet as well. Actually, the entire INIT packet does not have to be repeated and instead the client sends only a part of the INIT packet (e.g., receiver window, initial sequence number) required to recreate the TCB data on the server side just in case there is a cache miss. Later, when the COOKIE-ECHO packet arrives, first the SCTP server uses the index key to look up the corresponding TCB in the cache pool. If there is a cache hit, this TCB can be used directly as long as the Cookie is valid. Otherwise, a new TCB is created again using the repeated INIT information from the received COOKIE-ECHO packet.

To compare the overhead between the existing and the proposed Cookie mechanisms, the worst case scenario is assumed where the entire TCB data of 140 bytes is carried in the Cookie parameter for the existing case. The rest of the Cookie is the same for both cases and it takes up 16 bytes, because there are 4 bytes of the Type/Length fields plus 12 bytes of the Value field without the TCB part. In the existing mechanism, since the INIT-ACK and COOKIE-ECHO packets are exchanged along with the Cookie between client and server,

the total overhead comes to 360 bytes for the Cookie mechanism. On the other hand, our new mechanism does not send any TCB data in the Cookie parameter and only a part of the INIT packet is repeated in the COOKIE-ECHO packet. Because the repeated portion of the INIT packet is 16 bytes without the first Type/Length fields, the total overhead in this case comes to 96 bytes for the Cookie mechanism, which means about 73% reduction of overhead compared to the existing scheme, resulting in a significant improvement. Moreover, since cache hits are expected most of time, the setup delay can also be reduced substantially.

4 CONCLUSIONS

In this paper, we have presented simulation studies of SCTP in wireless Internet, with particular emphasis on the DoS resistance feature. Observe that SCTP achieves DoS resilience via its improved handshake mechanism and Cookie feature. We also have discussed the overhead issue in return of more security and proposed one possible approach to minimize it. We have performed detailed simulations to not only demonstrate the network's resilience to intruder attacks while using SCTP, but to also provide a detailed analysis of the associated overheads and delays with regard to SCTP's DoS resilience feature, and comparisons with TCP. Additionally our detailed simulation study has furnished invaluable insights into the feature and helped in the derivation of a novel Cookie mechanism that judiciously combines cache and INIT packet repetition to minimize the overhead and to maximize security at the same time associated with SCTP's DoS resilience over wireless Internet.

References

1. S. Aidarous and T. Plevyak, "Telecommunications Network Management - Technologies and Implementations," IEEE Series on Network Management, 1997.
2. L. Coene, "Stream Control Transmission Protocol Applicability Statement," IETF Internet Draft, November 2001.
3. D. Comer, "Internetworking with TCP/IP," Prentice Hall Publications, 1995.
4. H. Krawczyk, et al., "HMAC: Keyed-Hashing for Message Authentication," IETF RFC 2104, March 1997.
5. A. Law and W. Kelton, "Simulation Modeling and Analysis," McGraw Hill Publications, Second Edition, 1991.
6. M. Mathis, et al., "TCP Selective Acknowledgment (SACK) Options," IETF RFC 2018, October 1996.
7. R. Stewart, et al., "Stream Control Transmission Protocol," IETF RFC 2960, October 2000.
8. I. Joe, "Secure Routing with Time-Space Cryptography for Mobile Ad-Hoc Networks," Proceedings of IEEE MILCOM, October 2005.