University of Duisburg-Essen, Computer Networking Group

Security Evaluation of SCTP

Bachelor Thesis

Submitted by

Michael Nordhoff

Essen, September 2006

Abstract

The Stream Control Transmission Protocol (SCTP) is a new reliable transport protocol. Although first developed to transport telephone signalling messages over IP networks, it could be used as a general purpose transport protocol in future networks. Integration of data, speech and multimedia services into modern network topologies is a strength of SCTP compared to the traditional protocols. But crucial for the success of SCTP will be its robustness and security. Robustness versus malicious denial-of-service (DoS) attacks is essential to guarantee a high reliability and quality of service. In addition cryptographic services like authentication and confidentiality are required and must be provided for SCTP traffic in the future. There are already three important approaches, which all have their own advantages and drawbacks.

This work briefly explains SCTP and the security solutions. Further conceivable harmful DoS attacks are derived by analysing the SCTP standard. Tests were performed to evaluate SCTP's robustness in respect of different attack scenarios. Further on this work summarises the functional and performance-related drawbacks of the three security solutions. Then performed tests to evaluate the performance-related drawbacks – which are up to now theoretically deducted but not comprehensively confirmed and benchmarked with real test scenarios – are described and the results are discussed.

Contents

1 Introduction

The Computer Networking Technology Group at the University of Duisburg-Essen deals with the development of new and future communication technologies needed for computer and other modern communication networks. One field of study is the new transport protocol Stream Control Transmission Protocol (SCTP) with its applications and implementations. To establish SCTP as a multi-purpose transport protocol, the key criteria *robustness, security* and *performance* need to be taken into account.

SCTP features like the 4-way handshake mechanism without state retention at the server's side theoretically eliminate weaknesses of the traditional protocols like the Transmission Control Protocol (TCP). Quantitative tests may allow conclusions about the *actual* robustness and efficiency, when measuring the behaviour of an SCTP implementation under high load and denial-of-service (DoS) attacks.

A major problem of acceptance of the common SCTP is the lack of security. There are already different possibilities to provide authentication and data confidentiality for SCTP traffic. The standard security protocols IPsec and TLS in combination with SCTP are subject to functional and performance related limitations. To identify these limitations the standard security solutions are compared to an optimal solution, called Secure-SCTP (S-SCTP). This solution is assumed to be optimal, because the security functions are directly integrated into SCTP. There are two ways to compare the different solutions: on the one hand there are qualitative criteria. Security features, flexibility of usage, ease of usage, compatibility and other functional features can be compared in a qualitative way. On the other hand there are quantitative criteria: performance is one of the most important criteria to decide if a solution will be accepted or not.

To evaluate the performance and robustness of transport protocols in a realistic way, measurements should be performed in a real environment. But the conditions in a real network are constantly changing (e.g. cross traffic); hence it is not possible to compare data measured at different times. This work deals with measurements and results of measurements that have been performed in a testbed environment where control over all essential parameters – regarding the hosts and the network – is given.

2 Overview of SCTP

2.1 General overview

The Stream Control Transmission Protocol is a unicast transport protocol for IP networks that is standardized in RFC 2960 [1] by the Internet Engineering Task Force (IETF). It is located at the fourth OSI [2] layer like TCP and UDP and also uses IP as the underlying network protocol. It was developed because the existing protocols TCP and UDP could not provide functionality and features which are needed for current and future applications of IP networks (see chapter 1.3 of [3]). The initial development of the standard was performed by the SIGTRAN working group of the IETF to transport signalling data of telecommunication systems via IP-based networks. The task was to find a better protocol than UDP or TCP due to the needs of a better quality of service and more reliability. The result was a standard of a multi-purpose transport protocol which is defined in the RFC 2960 and is renewed by additional functions described in some RFCs and Internet-Drafts ([4], [5], [6], [7]).
Before shortly introducing SCTP, some SCTP-related technical terms have to be explained because their meanings differ from meanings in other contexts.

In SCTP communication parties are called *endpoints*. The communication relationship between these parties is called *association* and an SCTP *transport address* is a combination of an IP address and an SCTP port (see chapter 2 in [3]). SCTP has – like UDP and TCP – its own port number space. One endpoint can have more than one network interface, thus it can have more than one IP address, called *multi-homing*. An association with multi-homed endpoints can have multiple paths. A *path* is a determined, unidirectional network connection between two network interfaces of an association. One *message* is a coherent data unit with fixed boundaries that has to be sent from one endpoint to the other endpoint. A *chunk* is a formatted data block which is embedded in an SCTP packet. SCTP transfers data in one or more streams. A *stream* is a coherent sequence of messages or chunks respectively.

The rough format of a SCTP packet is as follows: it contains an SCTP common header and one or more chunks grouped behind the header. The common header consists of the source port number and the destination port number. These SCTP port numbers can – with the help of the IP addresses of the IP header – identify the association the packet belongs to. The next 4 bytes (32-bit word) contain the verification tag. It is allocated to the current association and prevents from confusion with old packages of former associations and also prevents blind attacks. A checksum builds the last 32-bit word of the common header. The checksum is calculated over the SCTP common header and all chunks. It guaranties data integrity. In the

original RFC it was defined as an ADLER-32 checksum algorithm, but nowadays it turned out to be a week algorithm and it was changed to a CRC32 algorithm [4].

Chunks are designed in a way that they are fully self-descriptive having a uniform format. There are two major types of chunks: data and control chunks. Control chunks are used to manage and control the association; to send SCTP-communication-related control information to the peer. DATA chunks carry the payload, i.e. the messages received from the upper layer protocol (ULP). SCTP can bundle control chunks and data chunks in one SCTP packet. In this case the control chunks must be located at the beginning of the packet before DATA chunks.

SCTP has got a set of essential features. Some of them have not been available in general-purpose data transmission protocols yet. Others are already well-known features of traditional protocols like TCP or UDP. SCTP is characterised by the complete set of provided features. Its main and most important features are briefly mentioned here:

- SCTP endpoints are able to have several network interfaces with one IP address each. It is called *multi-homing* (see chapter 2.2).
- Usually the messages within a stream have to keep the right order. An SCTP association can consist of several message streams; there is no dependency between messages of different streams. This feature is called *multi-streaming* and is also described in detail in the next chapter.
- Small Chunks can be put together in one SCTP packet. This reduces the overhead since the SCTP common header is just sent once. CONTROL and DATA chunks can be combined. *Bundling* is optional, the MTU has to be taken into account and some combinations are not allowed due to the logic of the association state machine. For further details have a look at chapter 6.10 of [1].
- *User data fragmentation*: SCTP takes the user data message-wise from the ULP. If the message is too big to fit into a single DATA chunk and SCTP packet with keeping the SCTP packet size below the MTU, SCTP fragments the message and puts it into more than one DATA chunk in different SCTP packets. With the help of the TSNs and the B and E ('begin' and 'end') flags of the DATA chunk parameters the peer is able to reassemble the message and pass it to the ULP like it was passed to SCTP from the sender's ULP.
- *Retransmission of lost packets*: SCTP is – like TCP – a reliable transport protocol and ensures that the transmitted data is really received by the peer. The Selective Acknowledge Chunk (SACK chunk) informs the sender about the TSN of the last DATA chunk that was received without any earlier missing chunks. This is called the 'cumulative ACK point'. Additionally complete sequences of received DATA chunks which were sent later than the cumulative ACK point are reported. Each sequence is defined by the TSN of the first and the last chunk. This SACK chunk information enables a very effective acknowledgement and retransmission algorithm.

- The *flow and congestion control* of SCTP is very similar to the TCP ones ([8], [9]). For flow control initially the values of 'receiver advertised window size' (a_rwnd) need to be exchanged. The sender keeps a parameter called 'receiver window' (rwnd) up-to-date. At the beginning of a session the rwnd value is set to the recently obtained a_rwnd. Sent data decreases the value, acknowledged data increases it. When rwnd goes towards zero, the sender stops sending. For congestion control SCTP uses mainly parameters called 'congestion window' (cwnd) and 'slow-start threshold' (ssthresh). These parameters are maintained for each path separately because different paths can be in completely different networks with very different behaviour. A transmission starts with a slow start phase. The capacity of the network is unknown and the transmission rate starts slowly; it increases – like the cwnd value – exponentially. The slow start phase ends when the cwnd value is greater than the ssthresh. The congestion avoidance phase follows; cwnd is increased by 1 packet per RTT. In case congestion is detected, it swaps to the congestion control phase, i.e. when packet loss is detected by the sender, cwnd is cut in half. This combination of the main rules is called Additive Increase Multiplicative Decrease (AIMD). In case the retransmission timer T3-rtx has expired, the cwnd value is set to one MTU and the transmission continues at the lowest level with the slow start phase. The same congestion control mechanisms for TCP and SCTP guaranties fairness for both protocols when they work together in the same IP networks and the Internet.
- SCTP *association management*: SCTP is a connection-oriented protocol; the association must be set up, maintained and shut down. For these purposes the procedures are exactly defined. It is also described in [1]; a concise state diagram can be found e.g. in [10].

Further introductions of SCTP can be found in the informational RFCs 3286 [11] and 3257 [12] and on our institute-maintained web site "SCTP for Beginners" [13].

2.2 Comparison to TCP

Basic features of SCTP have already be mentioned shortly in the chapter above. Now we are focussing on features constituting the difference between SCTP and the traditional transport protocol TCP:

- *Message stream versus byte stream:* A mayor difference between TCP and SCTP is that TCP transports a byte stream while SCTP transports messages in a stream. If a number of bytes are sent (via TCP) in one step and later some more bytes are sent, the receiver can not distinguish which bytes were sent in which step. The ULP of TCP must have a mechanism to recover the message boundaries. SCTP, in contrast, conserves message boundaries by operating on whole messages instead of single bytes. That means if one message of several

related bytes of information is sent in one step, exactly that message is received in one step. The traditional protocol UDP has already sent data message-wise, but it is not a reliable protocol; UDP sends every single message independent of the others and does not care about the sequence or if the datagram has reached the receiver. SCTP's manner of sending data – message-wise in a reliable association – leads also to other features and possibilities that have not been provided by TCP or UDP.

- *Multi-homing versus single-homing*: SCTP and TCP are both reliable transport protocols and use IP as the underlying network protocol. Both protocols need to initiate a kind of connection between the peer instances by performing a so-called handshake mechanism before transferring user data. Each TCP peer is linked with exactly one IP address. SCTP endpoints are able to have several network interfaces with one IP address each. During initiation of an association the SCTP peers exchange lists with their additional IP addresses. In case of correct implementation and configuration of two SCTP endpoints with X and Y numbers of IP interfaces, the associations can have X*Y different paths (pp. 26/27 in [14]). This feature makes it possible to obtain a high reliability and robustness against single interface and network failures. TCP connections have only a single path and break down when this path is not working any more. This has often direct consequences to the application layer: e.g. user sessions become invalid, application programs get interrupted or even terminated. An SCTP association has one primary path in each direction. The main load is transmitted over this primary path. In case of packet loss a backup path is used for retransmission. This avoids additional and unnecessary congestion at the primary path. Through the backup paths HEARTBEAT chunks are transmitted regularly to check the availability of these paths. During an association lifetime the endpoints always have a status of all paths. This is needed in case the primary path has a failure. In this case another *available* path can be chosen as the new primary path. The old path becomes unavailable but it is checked with HEARTBEAT chunks for reachability. Unlike TCP the SCTP protocol is able to keep the association alive in case of network failures. Load balancing is not yet available with SCTP multi-homing.
- *Multi-streaming versus single-streaming*: An SCTP association can consist of several message streams. Usually the messages within a stream have to keep the right order. If a message is missing or some messages arrive in another order at the SCTP peer, the peer has to wait until it can reorder the messages without missing any message before passing them to the ULP. There are many application scenarios where not all messages depend on other messages. They could be put into groups and only in these groups they are dependent and need to be in the right order. There is no need of an order between messages of different groups. Messages of different groups can be sent via different streams; all messages of a stream are ordered before they are passed to the ULP; but if a message is missing and the peer is waiting for it or for its retransmission, it doesn’t block the delivery of the other

streams' messages. They can be passed to the ULP, if they are in order. SCTP avoids the so-called head-of-line blocking, which occurs at TCP connections: here every association has a single stream of bytes. In case of a loss of an IP packet or an out-of-sequence delivery, TCP waits for the missing one and holds all 'younger' bytes in a resequencing buffer.

- *Flexible/unordered delivery versus strict ordered delivery:* If TCP segments arrive at the destination peer in the wrong order – i.e. the next segment of the sequence is missing so far – the other already arrived bytes need to be stored in the resequencing buffer. Because TCP is byte-orientated, there is no way to solve this head-of-line blocking. If SCTP DATA chunks of a single stream arrive at a peer in the wrong order and the ULP needs the messages in an ordered sequence, the other chunks – analogue to TCP – need to be stored in the resequencing buffer (see p. 38 in [14]). But in case a message does not need to be in the order of a sequence, a so-called U flag (unordered flag) can be set. In this case the peer passes this message directly to the ULP without putting it into the resequencing buffer. This avoids unnecessary blocking.
- *Cumulative acknowledgement versus selective acknowledgement*: The original TCP standard and implementation performs only a cumulative acknowledgement. The receiver does not acknowledge any actually received segments, which have a higher sequence number than a segment that is missing so far. This causes unnecessary retransmits and – in case of multiple packet losses – retransmission time-outs and significant throughput reduction caused by the congestion control. To avoid this lack of effectiveness, additional TCP options were introduced: TCP Selective Acknowledgement Options [15]. It provides the opportunity to acknowledge up to four isolated, but complete segment blocks that have a higher sequence number than the segment, which was acknowledged by the cumulative acknowledgement. SCTP provides actually the same feature, which is mandatory unlike the TCP option. Beyond this, it has not the restriction of only 4 segment blocks and thus may have better performance than the retrofitted and limited TCP implementations (Chapter 12.2.5 in [3]). Another problem of the TCP solution may occur when the connection was established while the server was under SYN flooding. Further details about this issue can be found in chapter 3.2.
- *4-way handshake versus 3-way handshake:* Both protocols specify a so-called “handshake” procedure where control messages have to be exchanged to setup the reliable communication. TCP uses a 3-way handshake with the SYN – SYN/ACK – ACK packet exchange. SCTP performs four steps to establish an association. The first three packets INIT – INIT/ACK – COOKIE/ECHO have very similar functions compared to the TCP handshake. The fourth packet COOKIE/ACK to acknowledge the reception and the correctness of the cookie is not provided by TCP although TCP options may also use state cookies. Further details about the state cookie can be found in the next paragraph. At first sight it seems that SCTP needs to send more control data during the initiation before user data can flow. But

SCTP actually can exchange data upon the delivery of the third packet. TCP also provides this 'piggy-packing' feature where user data can be sent with the third and all other ACK-flagged packets; but this is optional and not supported by all implementations and their APIs.

- Another difference, which occurs at the initiation of connections, is the *state cookie* which allows an SCTP server remain stateless during the beginning of the handshake. This protects against blind denial-of-service (DoS) attacks. Although current TCP implementations also have an additional option providing state cookies, it is not a mandatory function; some operating systems still do not provide it or have switched it off by default. From a cryptographic point of view the TCP state cookie is weaker (see p. 281 of [3]), and in case it is active, some TCP options and parameters are not supported any more. More details about this problem can be found in chapter 3.2.

2.3 Security solutions for SCTP

2.3.1 SCTP over IPsec

IPsec is an architecture to secure communication at the network layer, i.e. the IP layer. It is a suite of protocols described by the IETF in several RFCs. Defined are the architecture, base protocols, algorithms and the key management. The Authentication Header protocol (AH) offers authentication and integrity to all fields of an IP packet which do not change during transportation. The Encapsulated Security Payload protocol (ESP) offers confidentiality by encryption and authentication of the payload. Both protocols can be used in two different modes: the transport and the tunnel mode. In transport mode usually two communicating hosts are endpoints of the security associations at the same time. The IP header with source and destination address remains unchanged and unencrypted. In tunnel mode usually two networks are communicating in a secure manner. Endpoints of the security association are gateways. The inter-network communication is secure; the intra-network communication in each network is insecure. IP packets are completely encapsulated and even the original header can be encrypted between the networks. Each packet gets an outer IP header by the gateway. Security associations (SAs) are set unidirectionally, that is why usually IPsec communication needs at least two associations. Each SA consists of a security parameter index (SPI), the source and the destination address and is defined in the security association database (SAD) at both endpoints. A second database is called security policy database (SPD) and is used to map traffic to the different SAs.

The Internet Security Association and Key Management Protocol (ISAKMP) defines a framework for interaction between authentication, key management, and security associations

protocols. The Internet Key Exchange (IKE) protocol is in charge of negotiation between the endpoints to exchange symmetric keys needed by AH and ESP. Figure 1 shows a block diagram with the most important IPsec-related RFCs.

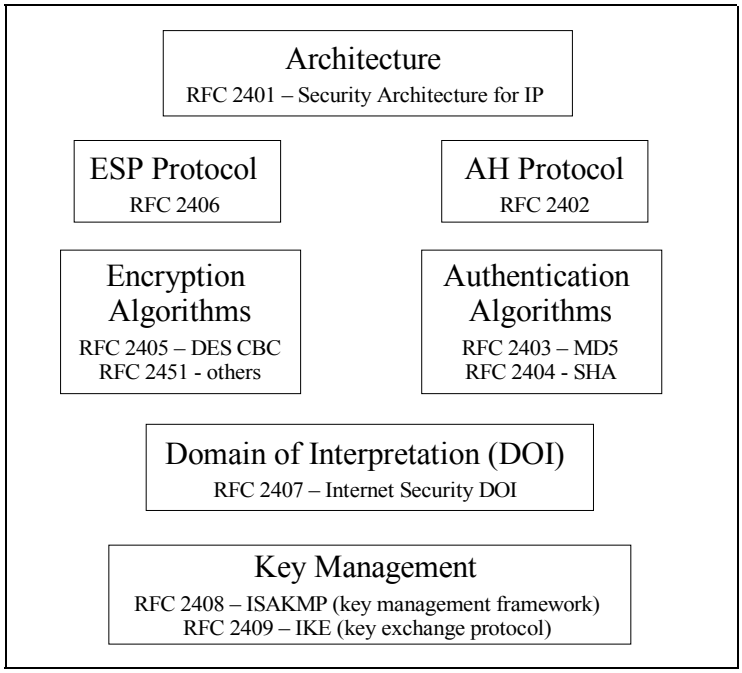

Figure 1: IPsec protocol and document overview [16]

It is possible to combine IPsec with SCTP. At first sight it seems to be straightforward to use IPsec instead of IP because IPsec is strictly limited to the network layer. The behaviour and interface to the upper layer – which is SCTP in our case – seems to be equal compared to plain IP. After setting up the SADs and SPDs at each endpoint SCTP runs with IPsec as it would run with IP. Problems occur with some features of SCTP: As a multi-homing protocol, SCTP holds a list with (one or more) IP addresses for each association endpoint. IPsec associations are defined with exactly one IP address at each endpoint. That is why key management for a SCTP association can become much more problematic compared to TCP over IPsec. The proposed SCTP feature of Dynamic Address Reconfiguration is not possible with IPsec. The document RFC3554 [17] defines the usage of SCTP with IPsec. It faces these problems with a proposed list of IP addresses for each SA's endpoint in the SAD. But at the moment there is no known implementation of this RFC. To run SCTP over IPsec with the conventional implementations, we need to manage the SAs individually for each path of a SCTP association.

2.3.2 TLS over SCTP

Transport Layer Security (TLS) is a protocol suite to add security features to the Transmission Control Protocol (TCP). It works at the upper boundary of the transport layer above TCP. TLS is the current name of the original labelled Secure Socket Layer (SSL). It was mainly developed to secure HTTP transmission over TCP. Later, several other application protocols (like SecureSHell, SecureFileTransmissionProtocol) were developed. As the interface of TLS to the application protocol is slightly different to the plain TCP's one, application protocols need to be adapted. The TLS suite consists of several protocols: the Handshake protocol, the Record Layer protocol, the Alert protocol and the Change Cipher Spec protocol (see figure 2).

The Record Layer protocol encapsulates all data send by TLS and takes care of fragmentation and reassembling of messages, encryption and decryption, authentication and verification using a Message Authentication Code (MAC).

The Handshake protocol is in charge of setting up a connection. It negotiates the cryptographic keys and algorithms for MAC and encryption. Its messages are also encapsulated and processed by the Record layer. Key management runs quite automatically although certificates (usually for the server) must be generated, authenticated and users need to be able to check them. The Alert protocol is used to send error or caution condition signals. Error signals can cause termination of the connection. The Change Cipher protocol informs the peer that the following packets are treated with newly negotiated ciphers and keys. 'Rekeying' needs to be performed.

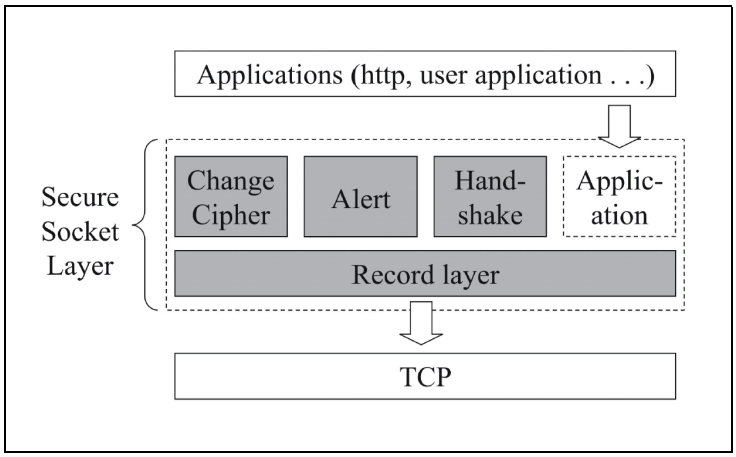

Figure 2: SSL protocol overview [14]

For using TLS with SCTP there is already a standard specified (RFC3436 [18]). Because TLS and SSL were developed for TCP, it requires a reliable, sequenced transport protocol beneath it. SCTP can provide this although features like unordered delivery or partial reliable

transport cannot be used. An advantage of SCTP is that many applications can use one SCTP association which reduces overhead of the network layer. Figure 3 shows a scenario where 3 applications use the same SCTP association. Each application uses one stream. Application number 2 uses standard SCTP. Application 1 and 2 use secured transmission by setting up a TLS session in their streams. The advantage is the possibility of mixing secured and unsecured data in one SCTP session in a flexible way. A disadvantage is a possible performance problem, in case a very high number of streams of an SCTP association need to be secured. In this case handshake and rekeying procedures, which need to be performed for each stream, can degrade the performance of the system. A single TLS session over more than one stream cannot be set up – due to the sequenced in-order requirement of TLS.

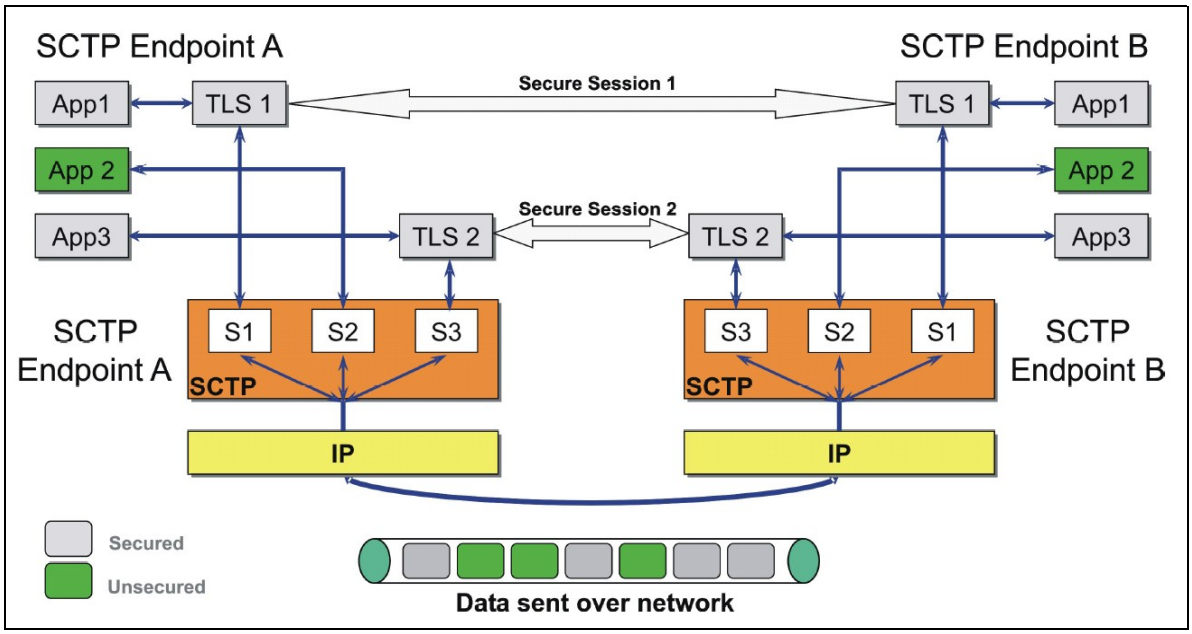

Figure 3: Secured and unsecured transmission using TLS over SCTP [14]

2.3.3 Secure-SCTP

The combinations of standard security protocols like TLS or IPsec with SCTP are subject to limitations, i.e. some essential SCTP features are not supported. That is why S-SCTP as an integrated security extension was proposed ([14], [19]). It is downward compatible to standard SCTP, has features to ensure authentication, integrity and confidentiality on a high security level, and should avoid performance problems. It is flexible e.g. with respect to mixing secured and unsecured data transmission.

The secure session is initialized after the normal SCTP association has been established. If it is not possible – due to one endpoint does not support S-SCTP or the setup of the secure session fails – the application can decide if it wants to use the unsecured association or if it shuts down the association. One S-SCTP association has only one secure session for all data streams in a multi-streaming case and for all addresses in a multi-homing scenario. In order to achieve this, the security mechanism is integrated between the upper functional block of SCTP which performs grouping of SCTP chunks to SCTP packets (bundling) and the lower functional block which performs the selection of network paths by choosing a destination address to send the SCTP packet as shown in figure 4.

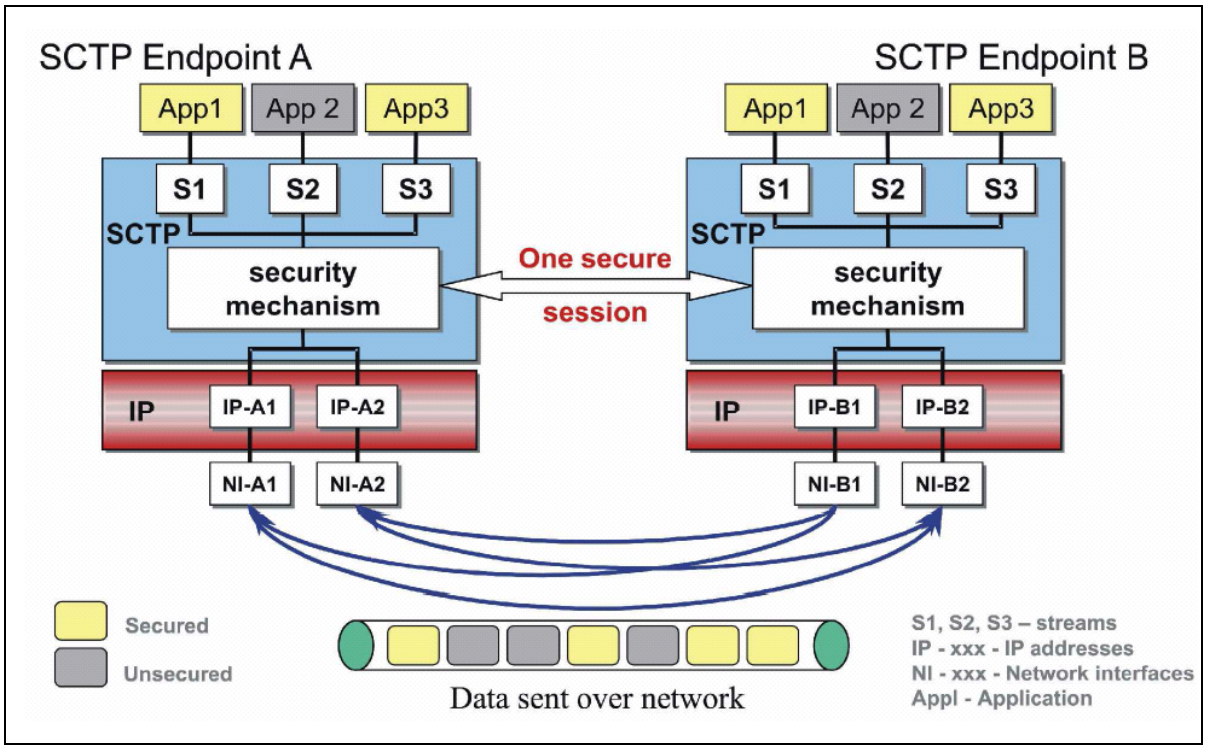

Figure 4: Secured and unsecured transmission using S-SCTP [14]

S-SCTP offers a set of security levels, which can be changed during a secure session life-time. An S-SCTP's performance disadvantage compared to TLS over SCTP may occur when long messages have to be fragmented at the SCTP layer. In this case S-SCTP has to secure each packet separately, so the overhead is bigger compared to TLS where the message is first secured and then fragmented. With respect to all other criteria S-SCTP should perform as well as or even better than the other two security solutions.

3 Denial-of-Service attacks

After the basic introduction of SCTP, we compared SCTP to the TCP and gave an overview of security approaches to improve SCTP. This chapter now deals with denial-of-service attacks. History has shown vulnerabilities of transport protocols, so we have a look at SCTP's behaviour related to this kind of attacks. In the following sections the motivation for this work is explained, subsequently the theoretical background is described, the measurements are presented and a conclusion is drawn.

3.1 Motivation and introduction

The SYN flooding weakness of TCP was first described in July 1996 in Phrack Magazine [20] and an exploit tool was given. In September of the same year first SYN flooding attacks were observed; an attack against mail servers of an Internet service provider (ISP) named Panix caused well-publicized outages [21]. This kind of attack was quite serious in comparison to other denial-of-service attacks – like exhausting network resources because with a SYN flooding attack the attacker needs less packets to get the same results.
Since that time countermeasures for limiting the impact of SYN flooding attacks were developed. On the one hand – which should only be mentioned for the sake of completeness – intervening routers were developed to filter these flooding packets; on the other hand TCP implementations were modified or extended to grant robustness for the communicating endpoints versus those attacks. SCTP as a new transport protocol and probable successor of TCP has to face these problems, too. The handshake mechanism is similar in a way that INIT flooding attacks could be performed analogue to SYN flooding attacks at TCP endpoints. SCTP was developed in the late 1990s when the protocol engineers were aware of attacks like SYN flooding. That is why countermeasures have been taken into account already in the early concept of the protocol. This should be an advantage compared to subsequent modifications and additional options of the TCP protocol. After pointing out existing and relevant protection mechanisms of TCP and the standardised mechanism of SCTP in a theoretical manner, this work will measure, show and compare the actual behaviour of the different protocols under attack with the help of measurements in a testbed environment. Because SCTP uses a state cookie mechanism to avoid state retention, also COOKIE/ECHO flooding attacks are imaginable. Parameters of the flooding packets (INIT and COOKIE/ECHO) should be changed to evaluate all possible attacks a malicious attacker would use. In the next paragraphs we will derive relevant test scenarios by analysing the theoretical background.

3.2 Theoretical background

Unprotected TCP endpoints are vulnerable related to DoS attacks. Systems providing services for the public Internet – like web servers and others – are listening for any connection setup acquired from any host with a valid (even spoofed) IP address. After the server has received a SYN packet, it allocates resources assuming that a connection will be established soon. The information about the future connection is stored in a data structure called Transmission Control Block (TCB). To avoid server's memory from being exhausted by connection requests, TCP implementations only allow a limited number of TCBs, a so-called backlog. If there is no response to the server's SYN/ACK by a client's ACK in a certain time, the connection information is deleted and the related memory is freed. During a DoS attack the backlog limit is reached. As the concept of the backlog is not standardised, the behaviour of different implementations vary. Systems could drop all additionally arriving SYN packets immediately as long as the backlog is filled up, discarding them silently or returning a reset packet (RST) is possible. Another way is to free an old TCB that is in the ACK-WAIT state and use the resource for the new connection. However, this behaviour harms the reliability of the server. Clients, that try to connect when an attacker performs a SYN flooding, do not get a SYN/ACK response and cannot establish a connection to the server. The attacker has just to send frequently a quick barrage of SYN packets with spoofed source addresses. The barrage must be large enough to reach the backlog. The source IP addresses must be unresponsive to make sure that no host answers with an RST to the victims SYN/ACK. This would free the related TCB immediately. If the source addresses of the SYNs are unrelated (especially not only the attacker's one), it is hard to detect a SYN flooding and furthermore hard to detect if a particular SYN packet is an attacker's one or a usual request. To keep the backlog filled-up, the frequency of the barrages must be chosen wisely by the attacker. The barrage must be sent when the lifetime of the former half-open connections exceeds and the TCBs get freed. As the timing might be hard, it is also possible to flood permanently although this kind of attack can be observed easily and network devises could limit or block it.

First ideas of mitigating the effects of SYN flooding were rising the backlog limit and reducing the TCBs' lifetime interval. Rising the backlog limit led to performance problems since the handling of the TCBs are not performance-optimised. Furthermore a change of the backlog size is only linear proportional to the amount of SYN packets an attacker has to send each barrage. Reducing the time-out duration of half-open connections improves the robustness only in a limited way, but causes unwanted time-outs for proper connection requests. Some implementations today additionally have SYN caches in use. In case the backlog limit exceeds, additional SYN packets are stored in a SYN cache. In this cache a half-open connection allocates significantly less memory. The SYN cache is developed for high performance even if the size is much bigger compared to the backlog. The SYN information is stored like

in a hash table. When there is memory free in the backlog, SYN data is read out of the SYN cache into memory. In case the SYN cache limit also exceeds, randomly chosen SYN data gets overwritten by the new connection details. Overwritten details of the half-open connection get lost, the connection cannot be established any more. A more common way to face SYN flooding is the SYN cookie mechanism. It avoids allocating state information at all for connections which are in SYN-RECEIVED 'state'. The state information is encoded in the initial sequence number (ISN) of the SYN/ACK packet. No information has to be kept in a TCB or SYN cache. In case the SYN was not spoofed, the client answers with an ACK packet. Its acknowledgement number – which is derived from the server's ISN (i.e. incremented by one) – is used to reconstruct the state information. Further on this information is stored in a TCB and the connection is established. The exact mechanism of encoding the state information into the ISN is implementation dependent. IP address and Port number pairs must be included as well as a kind of time stamp. The maximum segment size (MSS), which was given by the client, has to be integrated, too. Due to the limits of encoding into a 32 bit field (which has still to meet the TCP standards), not all possible MSS types can be encoded. Furthermore, additional TCP options can not be encoded. Although these options might become negotiated successfully by a SYN and SYN/ACK packet, the server does not save these items of information. Partly, negotiated options can be retrieved during a session – like a SACK option when a SACK comes from the peer. But in a common one-way data flow (from the server to the client) the server does not usually receive a SACK and therefore cannot use it. Like the SYN cache mechanism also SYN cookies do not allow piggy-packing with payload data [21]. Because of these disadvantages SYN cookies are often switched off by default (Linux 2.6.5 and earlier). Other implementations have hybrid solutions and combine the techniques of SYN cache and SYN cookies. FreeBSD 6.1 for example first uses a SYN cache when the backlog limit has exceeded. In case the SYN cache limit also gets exceeded, the SYN cookie mechanism is activated. More details about state cookies can also be found in [22] and [23].

Aware of the history of TCP SYN flood attacks and the approaches to mitigate them, the protocol engineers developed SCTP and provided it with a mandatory cookie mechanism (unlike TCP with its optional SYN cookies). The fields of an INIT/ACK chunk (figure 5) are equal to the INIT chunk apart from the additional state cookie, which is a variable-length, but mandatory TLV (type-length-value) parameter.

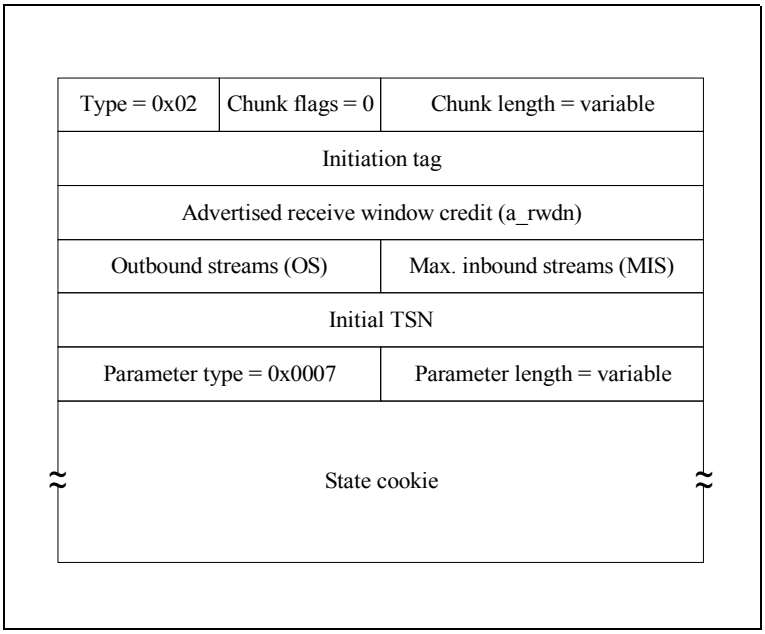

Figure 5: Minimal INIT/ACK chunk

In the strict sense the state cookie mechanism is optional and the state cookie could be empty (size=0) or unused. This would mean a total lack of INIT flooding protection and is therefore no issue. In RFC 2960 [1] the generation of the cookie is specified as follows:

> *The following steps SHOULD be taken to generate the State Cookie:*
>
> *1) Create an association TCB using information from both the received INIT and the outgoing INIT ACK chunk,*
>
> *2) In the TCB, set the creation time to the current time of day, and the lifespan to the protocol parameter 'Valid.Cookie.Life',*
>
> *3) From the TCB, identify and collect the minimal subset of information needed to re-create the TCB, and generate a MAC using this subset of information and a secret key (see [RFC2104] for an example of generating a MAC), and*
>
> *4) Generate the State Cookie by combining this subset of information and the resultant MAC.*
>
> *After sending the INIT ACK with the State Cookie parameter, the sender SHOULD delete the TCB and any other local resource related to the new association, so as to prevent resource attacks.*

These instructions are *SHOULD* declarations; the actual implementation is the developer's decision and not dependent on other implementations of the peers' SCTP instance because the content of the state cookie is exclusively meant for its own sender. Different kinds of MAC mechanisms – which have different cryptographic strength and different complexity – can be chosen.

To measure the robustness of SCTP and to compare it to TCP, the service quality of a server should be measured under attack. Issue of interest is the SCTP server response to INIT packets with the expected INIT/ACKs during INIT flooding attacks. These INIT floods could be performed by sending barrages with different amounts of INIT packets. A relevant value to measure is the rate of unanswered INIT packets. It is an important quantity of reliability as well as the response delay of the actually answered INITs, i.e. the round trip time (RTT) which can be measured at the client's side between the sending of the INIT and the reception of the corresponding INIT/ACK. Chapter 3.3.1 (“Basic DoS attacks”) deals with this kind of scenarios.

Another indicator of the service quality is the behaviour of an SCTP instance related to an already established association with user data transfer. When an INIT attack is performed towards the sending server, the remaining payload transmission rate should be monitored. These tests are described in Chapter 3.3.2 (“Established data transfer under attacker”).

3.3 Measurements

To evaluate the performance of transport protocols in a realistic way, measurements should be performed in a real environment. But the conditions in a real network are constantly changing (e.g. cross traffic); hence it is not possible to compare data measured at different times. To be able to compare the performance of the different scenarios and solutions, we need an environment where we have control over all parameters regarding the hosts and the network. This is the reason why we designed and implemented a testbed. In general the testbed developed in my foregoing work was used. Detailed information about it can be found in the documentation [24]. Especially for the tests described in chapter 3.3.2 the testbed has to be extended. Technical details can also be found in the appendix. All measurement results can be found on the enclosed compact disc.

3.3.1 Basic DoS attacks

In this test we want to find out the behaviour of SCTP during a DoS attack. Interesting parameters to get conclusions about the reliability of this protocol are the rate of unanswered

INITs (i.e. when there is no reception of the related INIT/ACKs) and the delay of the succeeded answers. With the same testbed TCP reference tests should be performed. For the basic DoS attacks the general hardware and network configuration of the developed testbed (see 2.3 of [24]) can be used without modifications. To get results that can be compared to the TCP measurements, multi-homing needs to be avoided. Hence, the second interface of each endpoint was shut down to avoid multi-homing. The sender should send single barrages of different size as fast a possible without bandwidth limitation. Traffic shaping at the router's system is not needed. One barrage has a size between one and 35000 packets. The INIT chunks were small and without any optional parameters (20 bytes). The flooding packets from the attacker have usual (i.e. not spoofed) IP addresses because it would not have any effect due to the stateless nature of half-open SCTP connections (from the server's point of view). It is important to suppress any COOKIE/ECHO or ABORT chunk that may return from the attacker to the server. As operating system FreeBSD was chosen because the release 6.1 provides a stable, well-supported and bug-fixed kernel implementation of SCTP. INITs were considered as not answered when no INIT/ACK arrived during the first second beginning with sending the INIT. Network audits have shown that there was actually no significant amount of INIT/ACKs responding later than one second. Each measurement with the particular barrage size was repeated ten times, and the (arithmetic) mean was determined. The comparable measurements with TCP were performed with Linux as operating system of the hosts. This guaranties a legitimate comparison of the cookie mechanisms. FreeBSD would perform its SYN cache mechanism or at least a combination of SYN cache and cookie mechanism. Unlike the cookie mechanism the cache cannot be switched off (Special tests with cache and cookie mechanisms of FreeBSD's TCP implementation can be found in [25]). Here the TCP tests were performed using the Linux (kernel 2.6.5) implementation of TCP with and without cookie mechanism. RST packets needed to be blocked by the attacker's own firewall 'iptables'.

In figure 6 and 7 the absolute unanswered INITs and SYNs are depicted. You can see that the loss of SCTP connections increases strongly between a barrage size of 100 and 1000 packets (please mind the logarithmic scale of the x-axis of figure 6, 8 and 10). With the further increasing of the barrage size the loss increases only very slightly, the curve is almost parallel to the x-axis in figure 7. The lack of the TCP answers with cookie mechanism is comparatively much lower, at all barrage sizes there were less than one hundred unanswered SYNs. By contrast the TCP endpoint without cookie mechanism ignores roughly proportionally the incoming SYN packets. Interestingly not more than 770 packets were answered in any of the tests.

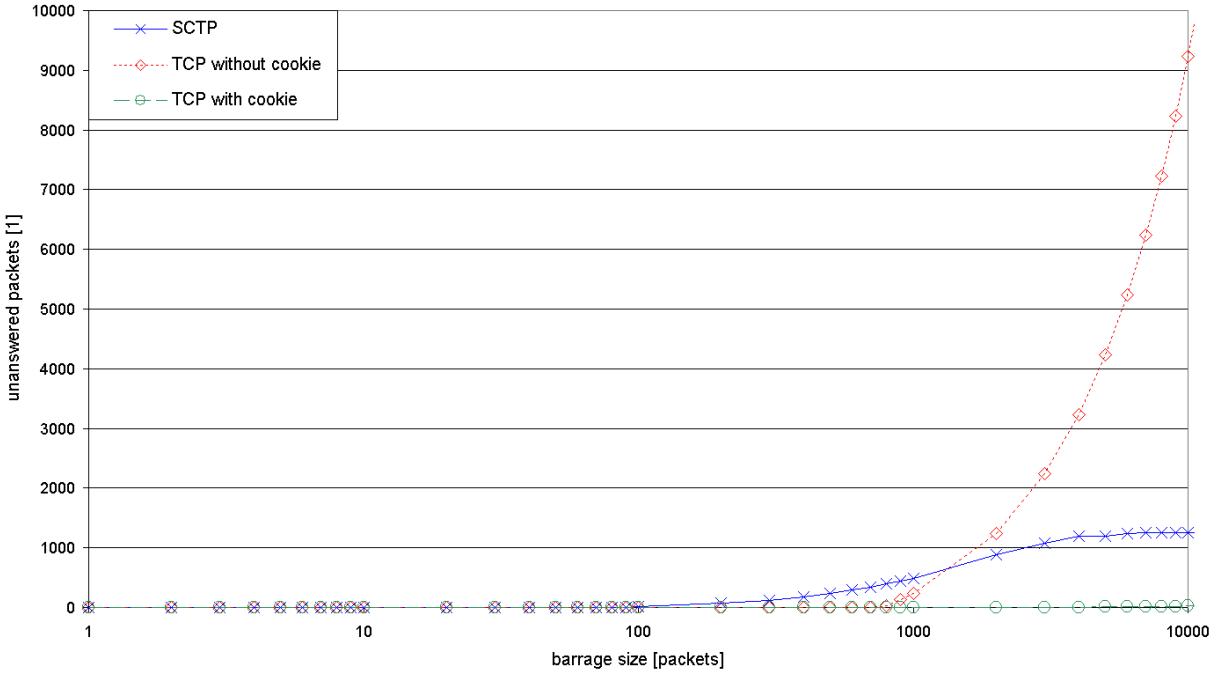

Figure 6: Absolute unanswered packets during INIT and SYN flooding – focus on small barrages

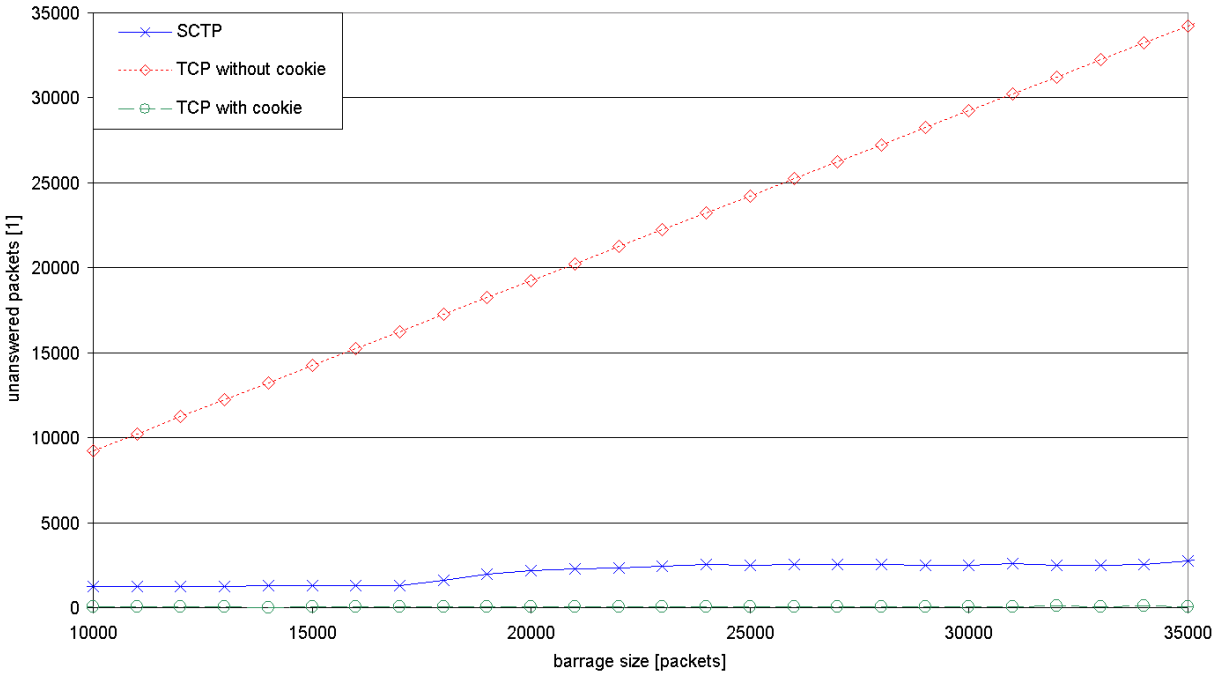

Figure 7: Absolute unanswered packets during INIT and SYN flooding – focus on large barrages

In figure 8 and 9 the relative rates of unanswered connection requests are plotted. The curve of TCP without cookies remains at 0% till a barrage of 700 packages and continues like an asymptote of the 100% limit. The loss rate of TCP with cookies is vanishingly low. The SCTP curve increases relatively early (first losses even at barrage sizes of less than 100 packets) and very fast even towards 50% at a barrage size of about 1000 packets. But with the subsequent increasing of the barrage size the loss rate lowers again and remains under 10 % since the barrage size is 12000 packets and higher.

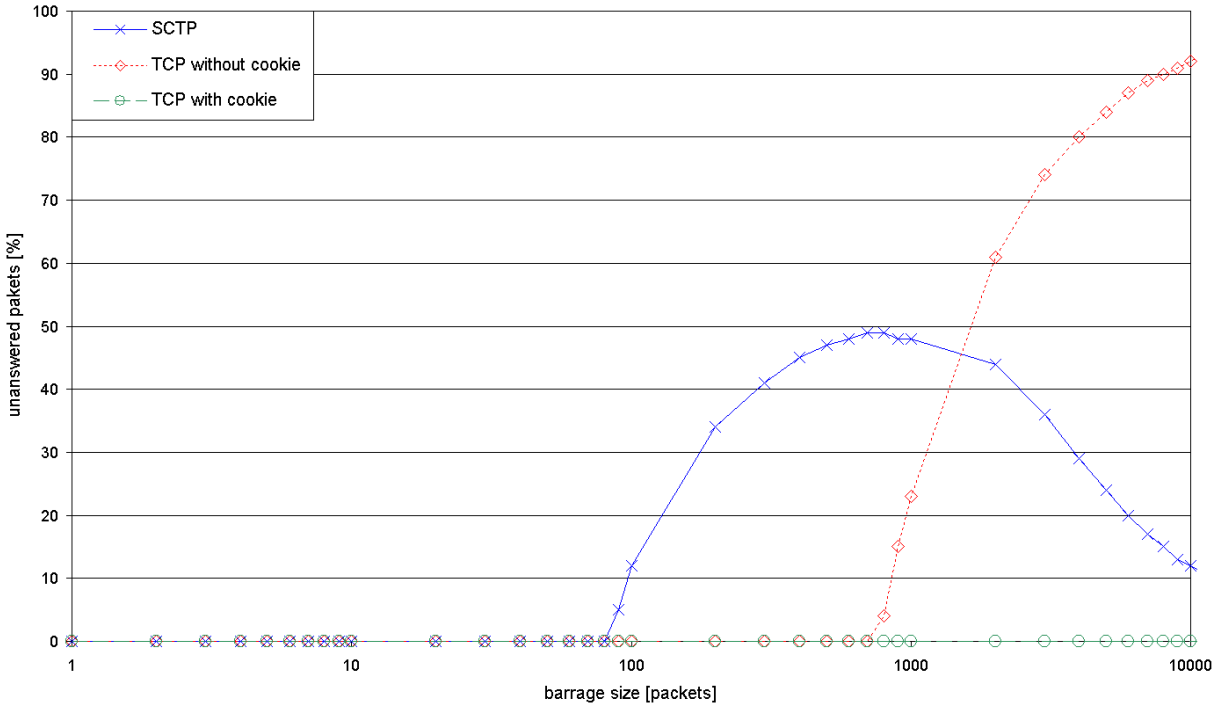

Figure 8: Rate of unanswered packets during INIT and SYN flooding – focus on small barrages

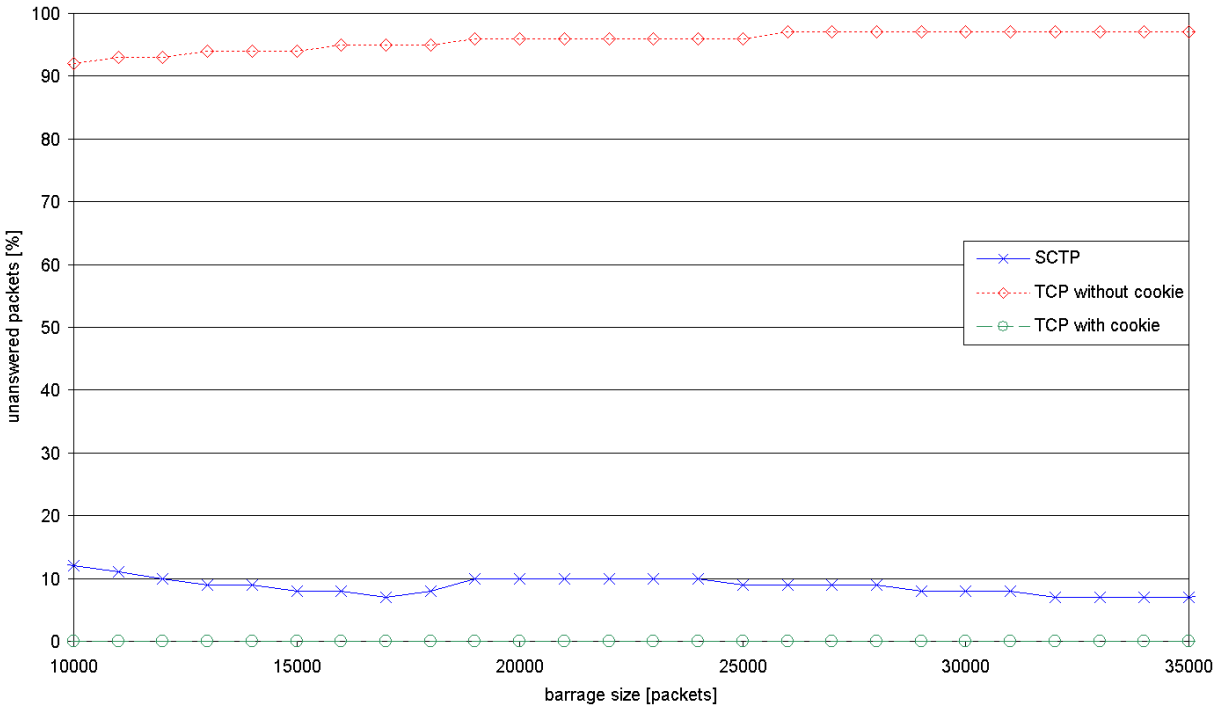

Figure 9: Rate of unanswered packets during INIT and SYN flooding – focus on large barrages

The round trip times were also measured at the attacker's host. The sent-times of the INITs or SYNs were subtracted from the arrived-times of the related INIT/ACKs or SYN/ACKs.

SCTP's RTTs increase fast at the lower barrage sizes (see figure 10). For example the average RTT at 1000 packets is about ten times as high (namely 3451 microseconds) as the average RTT of a single packet (328 microseconds). But further on the RTT decreases again and settles roughly around 2000 microseconds even at the barrage size of 35000 packets (figure 11). The RTTs of the TCP tests are much shorter. It does not increase significantly when rising the packet size and remains most times under 500 microseconds, i.e. the delay of SCTP's INIT/ACK is more than four times as large as TCP's SYN/ACK delay – both with their own cookie mechanisms. It has to be mentioned that the curve of TCP without cookie is not very meaningful; we have to keep in mind that only a vanishing low rate of packets were answered at all.

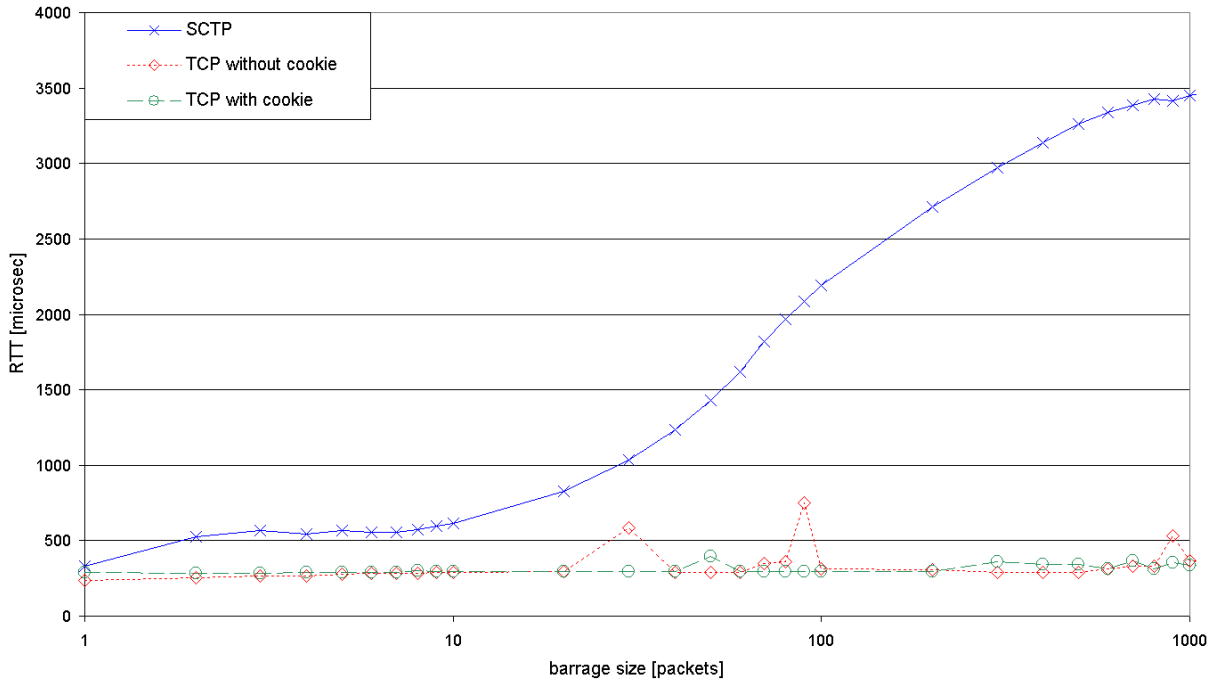

Figure 10: RTT of answered packets during INIT and SYN flooding – focus on small barrages

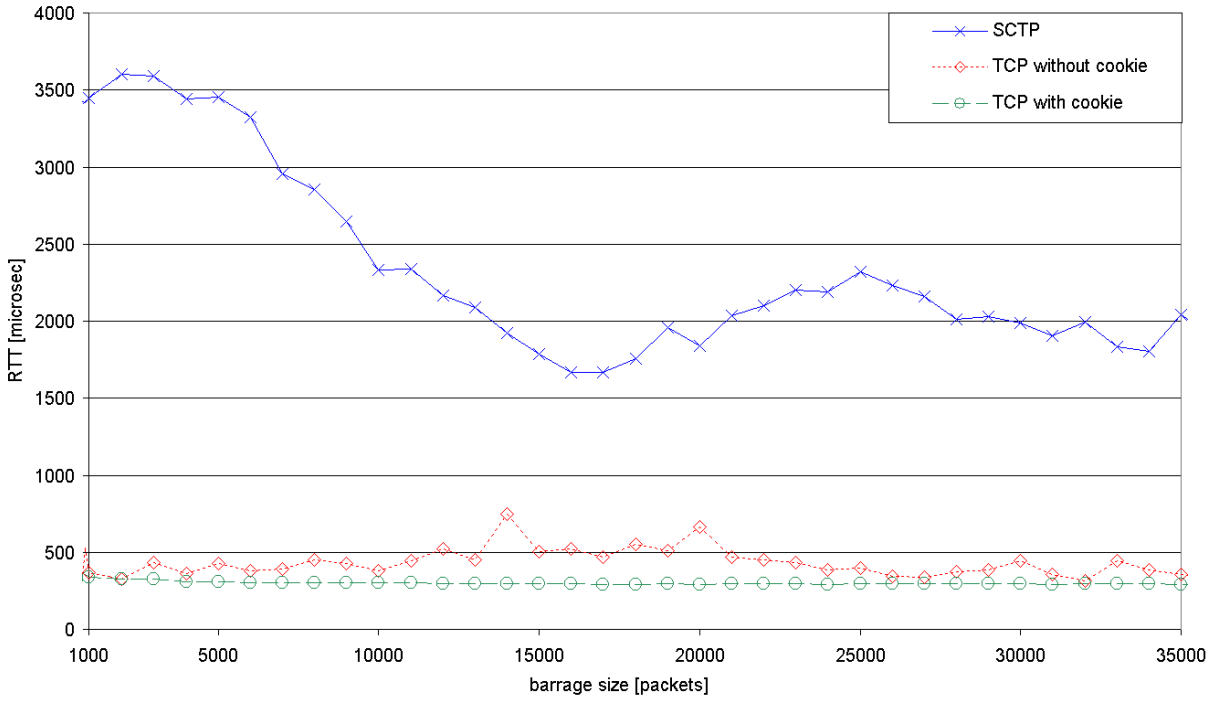

Figure 11: RTT of answered packets during INIT and SYN flooding – focus on large barrages

3.3.2 Established data transfer under attack

The next measurements also deal with the robustness of the SCTP implementation during DoS attacks. But now the quality of service should be evaluated by analysing the behaviour of a communicating endpoint that sends data using an already established association. During the constant data transfer malicious DoS attacks should be sent to the data-sending host and the remaining throughput to the client should be measured.

For this scenario the known testbed (used for the tests described in 3.3.1) can be used although it has to be modified, i.e. it must be extended. Additionally to the *server* and the *router*, we need two different endpoints that can communicate with the server: the *client* has to establish an association and receive data from the server; the *attacker* performs DoS attacks. Both, client and attacker, are connected at different router interfaces in different networks. (See figure 12.) The server is (apart from the router) the only multi-homed host: it has two interfaces connected by two different networks with the router. The routing tables can be adapted for using a multi-homed or a single-homed server. During SCTP tests we can choose wether the server is connected to the client and the attacker using the same interface (thus usual traffic and malicious traffic use the same interface of the server) or wether the connection to the attacker is given by a completely separate (backup) path with its own interface.

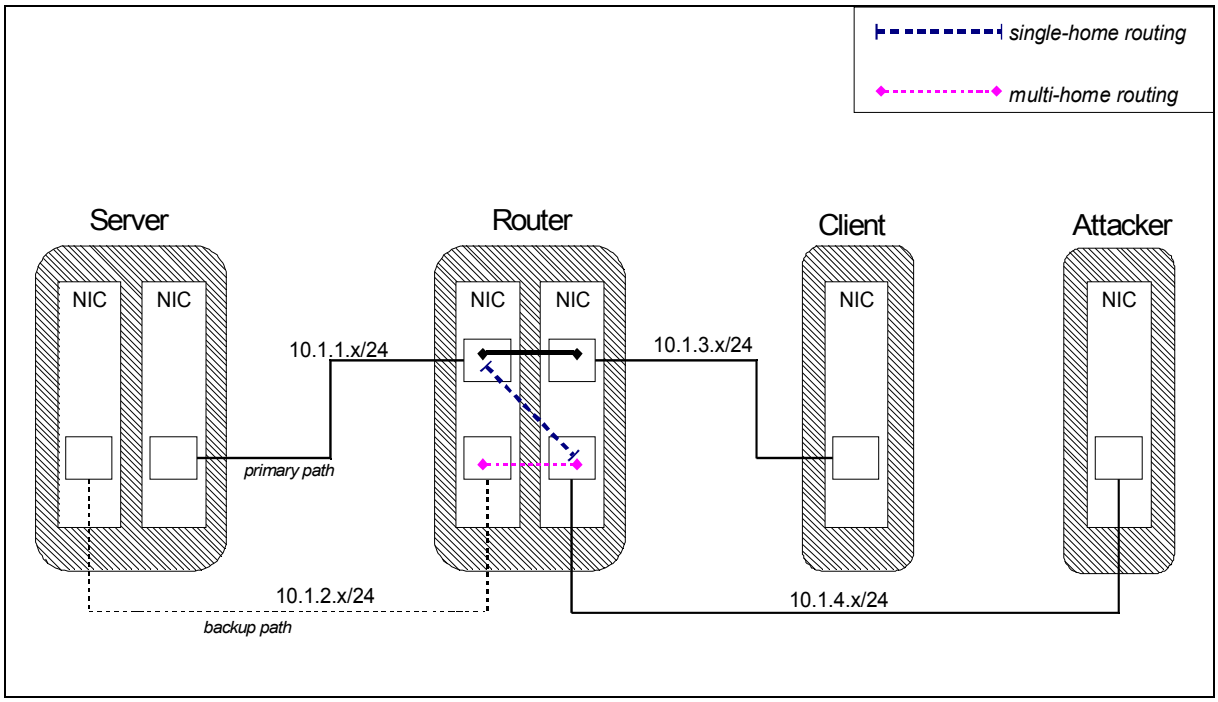

Figure 12: Testbed configuration with different routing for single-homed and multi-homed server

With the help of this testbed we try to simulate a realistic scenario. The Internet is an open network where the access is public and malicious attackers can cause troubles. Especially servers that are accessible via Internet must face undesirable traffic. Typically for the Internet are links with very different bandwidth limitations. For example a home user might have an Internet access of a very small bandwidth; others might have broadband access. Also the links inside the Internet vary widely. Consideration these facts we use a traffic-shaping software installed at the router. The software *dummynet* [26] meets all the requirements; it can pipe the routes separately and applies bandwidth limitations, delay and loss to it, i.e. in our testbed we can shape the inoffensive traffic (between server and client) and the malicious traffic (between server and attacker) separately. Further details about the configuration of *dummynet* in this testbed can be found in [24].

Due to the stateless nature of SCTP's half-open connections DoS attacks can only try to exhaust the limits of CPU performance and – mentioned here for the sake of completeness – network performance. To gain the best results – from the attacker's point of view – we have to find the most effective way to exhaust the victim's resources with limited means, especially with limited bandwidth towards the victim. We have to consider different opportunities to exploit possible vulnerabilities of the handshake mechanism. First we can send common INITs (analogue to the tests in 3.3.1). In this case the victim has to response with an INIT/ACK to the origin (or assumed origin) of the INIT. Additionally we can try to exhaust the victim's resources by sending INITs with unusually long parameters. There is no strict limit of the INIT chunk size, in general any kind of parameter of the TLV is possible (even the type is not known by the recipient). The victim has to react to extra-long INITs (LONG-INIT) in a proper and defined way – without getting exploited. The response of an INIT must always be an INIT/ACK with a cookie that contains all data to reconstruct the received INIT chunk – including the large parameters.

Another possibility to examine the robustness of SCTP is to send floods of COOKIE/ECHO packets to the server. The attacker-generated (and therefore invalid) cookie inside the chunk can have random data. Every time the victim receives such a packet, it has to check if the cookie is originally generated by itself. After realising that the the cookie is invalid, the receiver has to discard it silently.

A more sophisticated attack could be sending a COOKIE/ECHO with a valid cookie. This can be done by starting the attack with the first three parts of a usual 4-way-handshake and further on replaying the COOKIE/ECHO with the real cookie earlier obtained by the victim. In this case the victim first verifies the validity of the cookie. If the lifetime check is also positive, it has to realize that another association is already established. With the help of the tie tags it has to exclude all possibilities of other reasons of the replay (peer restart, setup collision, lost chunk) and finally has to discard it silently, too. (See more details in chapter 4.7.2.3 of [3]). A disadvantage of this kind of attack from the attacker's point of view is the

mandatory revealing of his actual IP address. The attacker has to send the INIT with his correct source address to enable a subsequent receipt of a valid cookie. This fact could be used to detect attackers by intrusion detection systems (IDS).

Another scenario could be that all the above mentioned attacks are run towards an SCTP port which is actually closed. In this case the server has to generate an ABORT chunk and send it back to the (assumed) origin.

Now we have a set of scenario parameters for the following tests. The aim is to evaluate at which circumstances the SCTP implementation is more robust and which scenario configurations might reveal vulnerabilities.

Table 1 shows all parameters of the test scenario that can be changed. With the help of low-precision pre-tests the dimensions of the results could be roughly estimated, it was especially helpful for choosing appropriate bandwidth limits. Additionally, we realized that some bandwidth limits (marked with a '*') could not be utilised completely by the attacker in some cases, due to the attacker's performance limitations. The actual bandwidths in these special cases are mentioned later. In general, the tests were done with all combinations of the parameters, which are theoretically about more than two hundred. Actually the amount is a little bit lower due to some tests which cannot be performed: the REAL-COOKIE attack can not be performed at all towards closed ports because the attacker does not get a valid cookie from the victim.

parameter	***options***
link bandwidth of 'good' data	– 100 Mbps – 50 Mbps
link bandwidth of attacker's connection	– 1 Mbps – 2 Mbps – 4 Mbps – 8 Mbps – 16 Mbps * – 32 Mbps * – 64 Mbps *
kind of attack	– INIT flooding (20 byte chunk) – LONG-INIT flooding (1064 byte chunk) – RND-COOKIE/ACK flooding (104 byte chunk) – REAL-COOKIE/ACK flooding (104 byte chunk)
server's connectivity	– single-homed (attack on primary interface) – multi-homed (attack on secondary interface)
attack's destination port (server-side)	– same as inoffensive data association – closed port

Table 1: Test scenario configuration parameters

The server was running on FreeBSD (release 6.1) with the current SCTP kernel patches obtained by R. Steward. A simple SCTP server – using the SCTP-API – was written: it listens to a single port, after the connection to the client has been established it starts sending an unlimited amount of 1000-Byte data chunks as fast as possible (apart from congestion and flow control). As payload data, a fixed byte stream (like one thousand times 'a') was used to save CPU time. The client – using the same operating system and SCTP stack – started initializing an association to the given SCTP peer. The buffer was read out without delay, the received data was counted, after every second the result was printed (or written into a file) and the counter was set to zero again. The attacker needed a more sophisticated program which was able to generate exact packets independent of any association state, protocol standard or API. With the help of SCTP testtool (STT) [27] it was possible to generate and send the needed packets.

The procedure of each test was as follows: First the network (interface configuration and routing table) and dummynet had to be set up. While the server was running, the client started. It initialised the association, the server started sending the data chunks and the client started monitoring the throughput. The next 30 seconds this data transfer was performed without any attack, to make sure the connection is in steady state. Exactly after 30 seconds the flooding attack was started by the attacker host, which lasts exactly 60 seconds. To find out the behaviour of the communicating endpoints after the attack, the connection was kept another 30 seconds while monitoring the throughput as well. After altogether 120 seconds this single test was finished, the monitoring client and the server were terminated, a certain time was waited before starting the next test to make sure all resources were freed again, the dummynet pipes were empty and the network was completely calm. Every single test was performed ten times to check the reliability and significance of the outcomes.

In this work I will display only a small amount of results. Only the meaningful outcomes are shown in the selected figures. To gain meaningful curves which represent the result of the measurements, I decided to display the median of the series of measurements in all charts of this chapter. It is more robust in the presence of outliers than the mean. Due to a few unpredictable and inevitable values which were close to zero even before starting the attack, the mean would show confusing dents by values that are far away from any actually measured values.

Figure 13 shows the results of the following configuration set: the server was single-homed, the flooding attack was run towards the same interface, that was also used for the data transfer to the client. (See again figure 12.) Client and attacker used the same open port of the server. Dummynet limited the link bandwidth between server and client to 100 Mbps (megabit per second) – like a Fast Ethernet link – and 50 Mbps link to simulate a network path with more congestion leading to a lower utilisation of the server. All denoted links and their bandwidths are meant as full-duplex. In this figure INIT flooding attacks with 20-byte INIT

chunks are compared to so-called LONG-INIT attacks with their STT-generated chunks of 1064 bytes. You can see that the curve of “INIT 100/8Mbps” denotes a short slow start during the first two seconds. Further on the transfer rate continues in a steady state at 91 Mbps; the link utilisation seems to be maximized. After starting the attack at 30 seconds, the transfer rate goes immediately down towards 13 Mbps and continues at that level until the attack quits at 90 seconds. Here, the transfer rate again goes immediately up to the former level of 91 Mbps. The results of the same test with 50 Mbps as inoffensive data transfer (“INIT 50/8Mbps”) are comparable. The payload throughput is cut in half (46 Mbps instead of 91 Mbps) like the link limitation (50 Mbps instead of 100 Mbps). During this attack (which is completely the same) also the remaining user data transfer rate is the same (13 Mbps). Here, the data transfer rate is not an objective of the link limitation towards the client any more, rather it is a result of the server's utilisation caused by the attacker. We have to keep in mind that the attacker's INIT floods and corresponding server's INIT/ACKs do use the same physical link between server and router but are *at no time* object of the link (or better: path) limitation between server and client controlled by the router with dummynet. The end-to-end link between server and router is a Gigabit Ethernet link and therefore no bottleneck, even for the sum inoffensive and offensive traffic. Both LONG-INIT curves settle roughly at 40 Mbps during the attack with a small, but not meaningful difference. Compared to the INIT curves which are at 13 Mbps the LONG-INIT flooding attacks have less effect than the usual INIT flooding attacks in case the attacker's bandwidth is fixed.

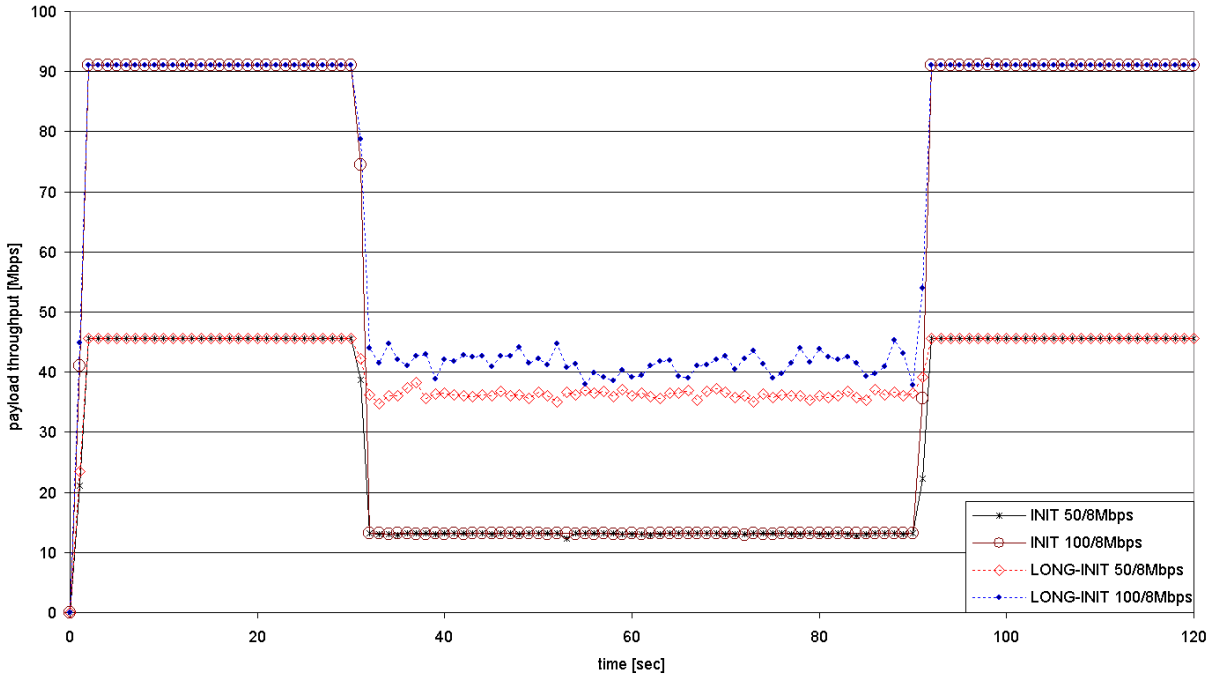

Figure 13: INIT versus LONG-INIT flooding attacks

After analysing the INIT and LONG-INIT flooding attacks, the RND-COOKIE attack should be focused. During this attack the attacker sends floods of SCTP packets with COOKIE/ACK chunks – which seem to be responses of a reputed but actually not received INIT/ACK – towards the server. The mandatory cookie in this chunk is just a random-generated field of data. The chunk size is 104 byte – like an authentic COOKIE/ACK chunk. All other configurations are exactly the same as mentioned earlier in this chapter. But the results are different: there was no measurable reduction of payload throughput during the attack. The server system was stressed so less that it could transmit the client data fast enough to fill up the links. The attacker could manage to send RND-COOKIE flooding attacks of 30 Mbps when there was no bandwidth limitation. Even during these attacks the server could not be forced to reduce the payload transmission to the client. Figure 14 shows this result compared to the former described INIT attack.

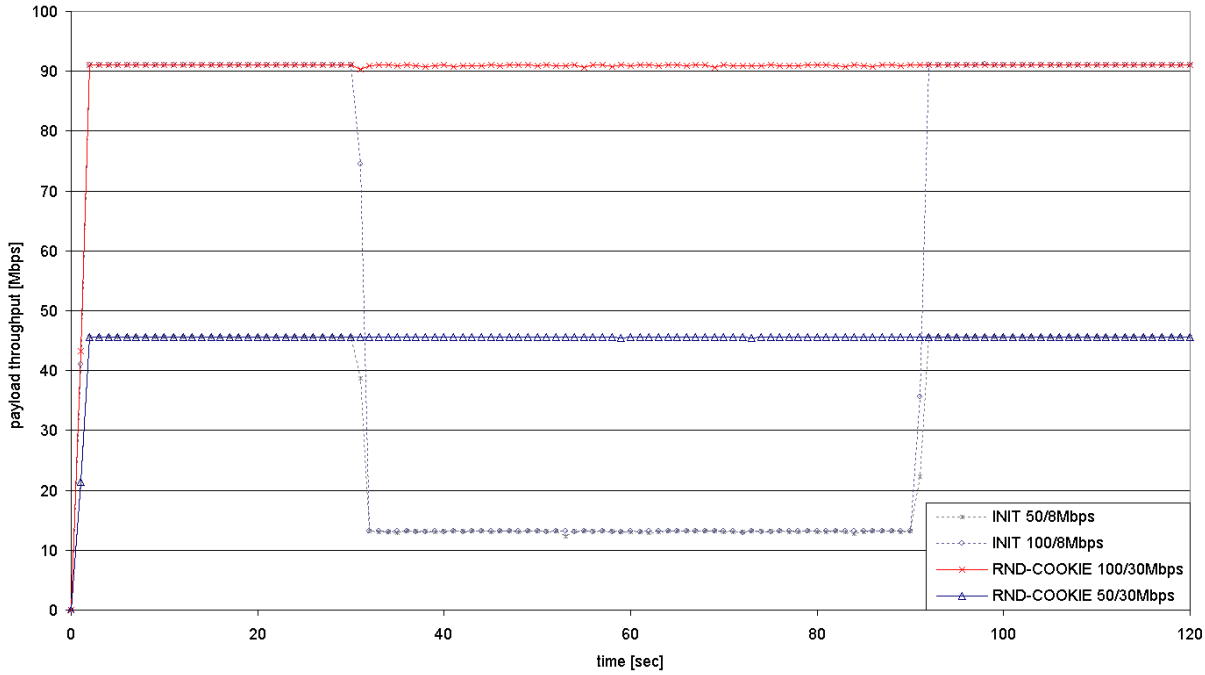

Figure 14: RND-COOKIE attacks compared to INIT attacks

The more sophisticated way to attack with COOKIE/ACK chunks is to use a valid cookie. Before starting this kind of attack, the attacker sends a non-spoofed INIT to the victim and uses the obtained cookie of the INIT/ACK chunk. Like all attacks described in this chapter the REAL-COOKIE flooding attack was performed by using STT, too. The results are displayed in figure 15: the REAL-COOKIE attack does not harm the victim either if the attacker's bandwidth is limited to 8 Mbps. Otherwise when the bandwidth is not limited and the attacker can flood as fast as possible – and this was up to 32 Mbps – the victim is less robust than during the RND-COOKIE attack. Nevertheless the REAL-COOKIE attack is still significantly less harmful than the INIT attack which is mapped again in this chart as reference.

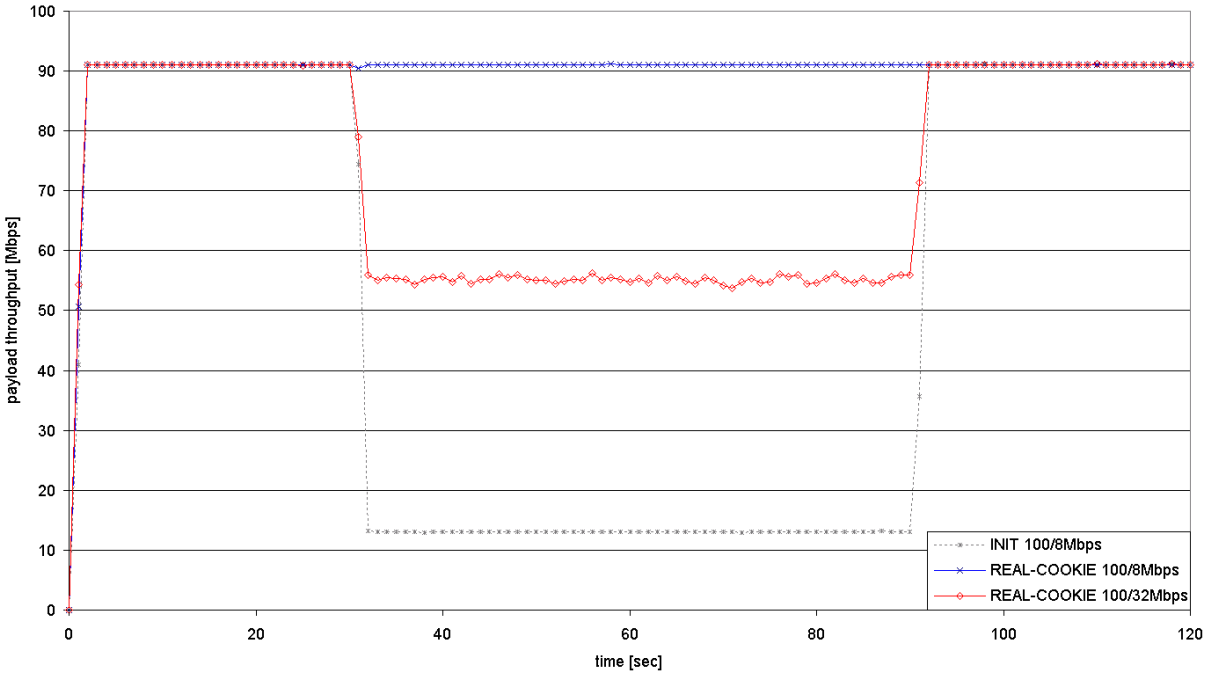

Figure 15: REAL-COOKIE attack compared to INIT attack

We have compared attacker's strategies with different bandwidth limits but constant testbed configuration. A further step is to change the testbed configuration. It is imaginable that an attacker directs flooding attacks towards a server's interface which is actually not used as the primary path of the user data transmission. If the server is multi-homed, the attack could impinge at the backup interface. For this reason we set up the second interface of the server, which is also connected with the router, although it runs in a completely different network. The routing tables need to make sure that the attacker reaches the server via this backup interface and also the server responses (in case it has to) to the attacker using the same one. The inoffensive data still uses the same path as before. The SCTP endpoint of the server is now listening also at the second interface with the open port. Figure 16 shows the comparison between an attack towards a single and therefore primary interface and an attack run towards the server's backup interface. This example displays the INIT flooding attacks with the certain bandwidth limitations used before. Obviously, the SCTP server is in this case more robust against INIT attacks arriving at the backup interface: the payload throughput does not fall under 35 Mbps unlike the single-homed case we have not more than 13 Mbps. I also ran the other kinds of attacks (LONG-INIT, RND-COOKIE, REAL-COOKIE) against the backup interface, the results were according to the measurements with a single interface, but over all the throughput level during the attacks remained higher.

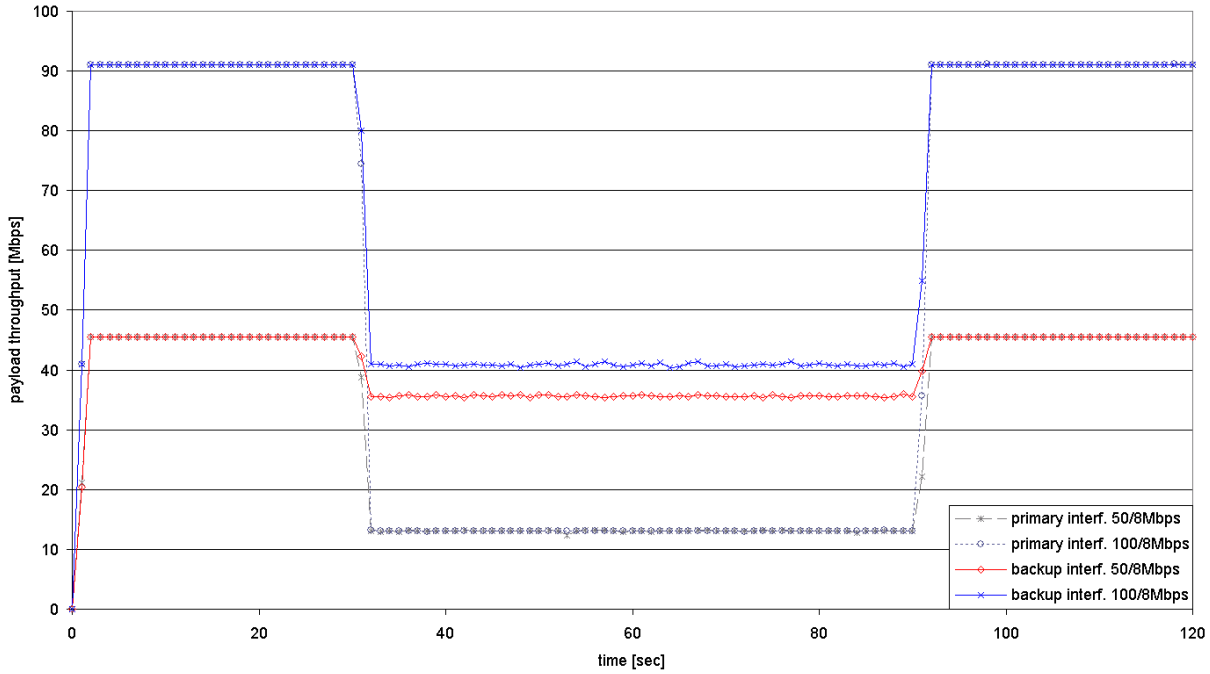

Figure 16: Attack against primary interface versus attack against backup interface

The only parameter which has not changed up till now (see table 1) is the state of the port to which the attacker addresses the floods. The former tests have been run towards the same port which was also used for the data transfer to the client. Due to the server program it is open and listens to any incoming connection requests. At this point we want to find out the server's robustness during attacks towards closed ports. In this case the server has to generate an ABORT chunk (with the length of 4 bytes) and send it to the attacker. First I reconfigured the server to run with only one interface again. The network and routing tables were also reconfigured. Then the attacker simply used another destination port number for the flooding attacks. Again all kinds of attacks were performed. For representing all results again the INIT attack is chosen. Figure 17 shows the throughput under attack; the configuration was again the single-homed server (i.e. the attack at the only and primary interface) and the bandwidth limitation as used before: 8 Mbps for the attack and 50 and 100 Mbps for the server-client link. The results are clear: an 8 Mbps INIT attack cannot harm the inoffensive data transmission if it is directed towards a closed port. After removing the link limit, the attacker performed a 14 Mbps attack. Under that kind of attack the payload throughput of the 50 Mbps link did not change, but at the 100 Mbps link it was reduced from 91 to about 78 Mbps. Nevertheless this attack has still much less effect according to the same attack towards an

open SCTP port. The measurements with the other kinds of attacks towards the closed ports produced comparative results: Flooding attacks towards closed ports are harmless or even without any effect. A further example is displayed in figure 18: 16 Mbps LONG-INIT attacks towards a closed port do harm the payload transmission to the client, but this attack is significant weaker than the corresponding LONG-INIT attack towards the open port.

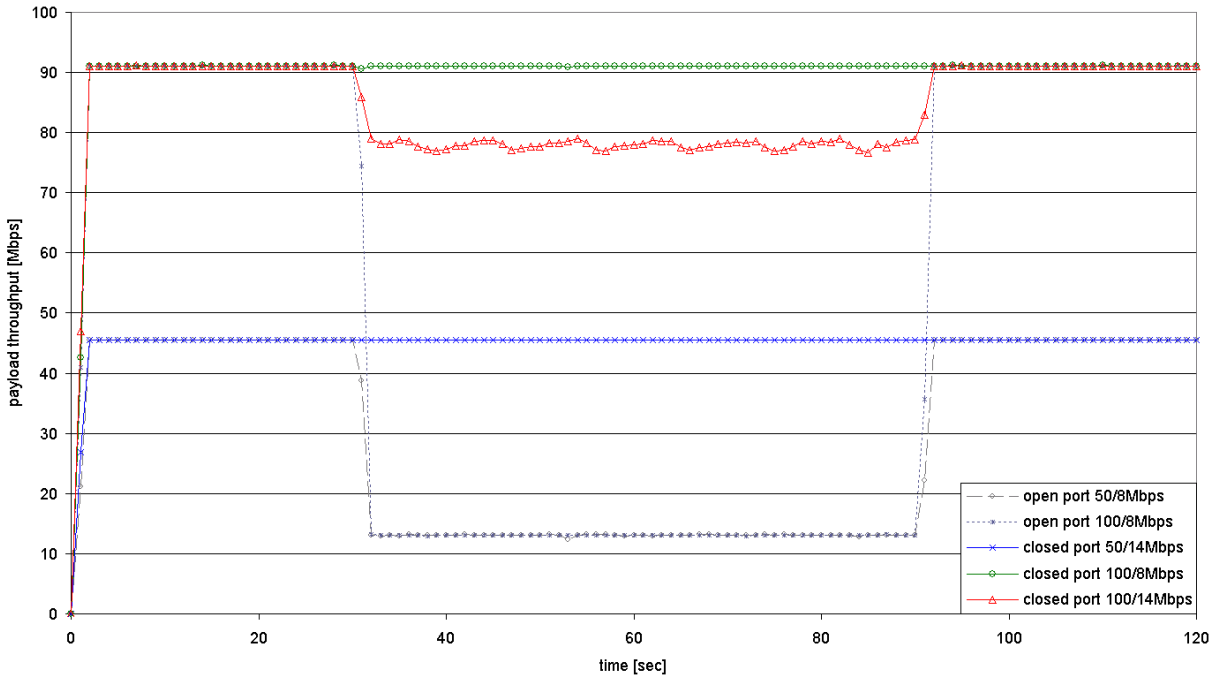

Figure 17: INIT attacks towards open ports versus INIT attacks towards closed ports

The last test configuration which should be mentioned here is an attack towards a closed port of the backup interface. All three possible kinds of attack were performed. The results are not shown in a figure because there are simple enough to describe in words: Non of all possible tests with the maximum strength (INIT: 8 Mbps; LONG-INIT: 64 Mbps; RND-COOKIE: 30 Mbps) have shown any effect concerning the server's data transmission to the client.

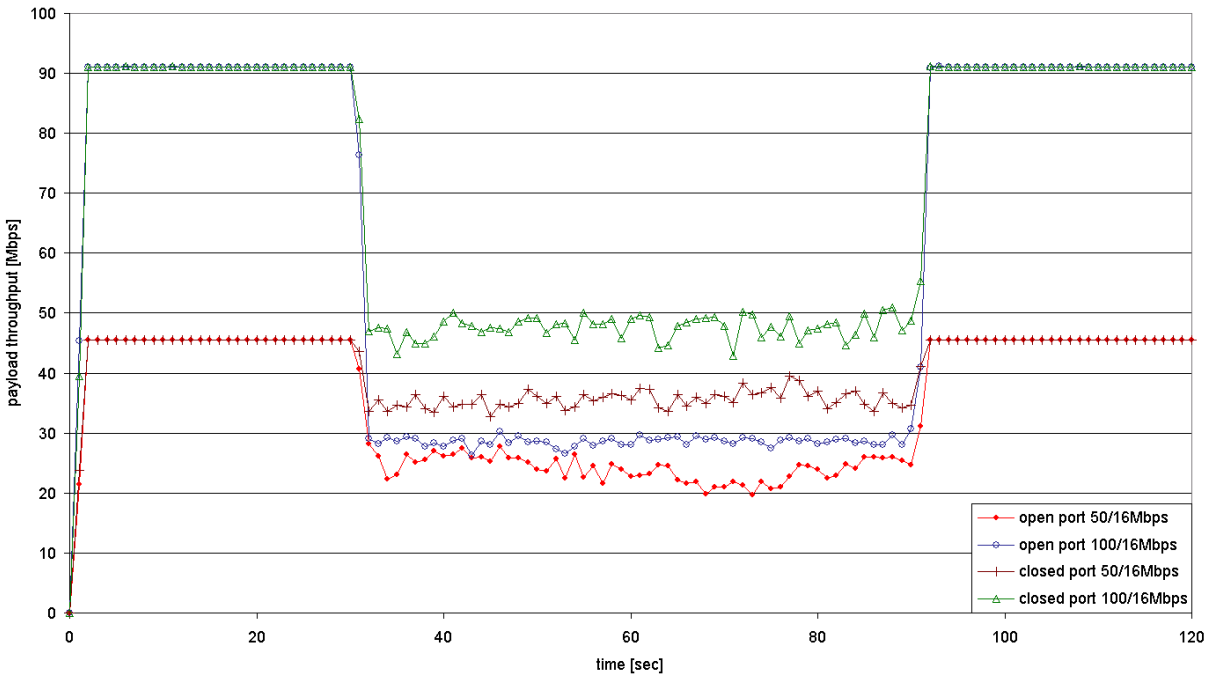

Figure 18: LONG-INIT attacks towards open ports versus LONG-INIT attacks towards closed ports

To classify and evaluate the behaviour of SCTP with the results shown above in a wider context, reference measurements with TCP implementation are imaginable. The aim is to compare the most effective attack towards the SCTP handshake mechanism with the corresponding attack towards the TCP handshake mechanism; i.e. the INIT flooding attack should be compared with the SYN flooding attack. A comparable configuration set has to be chosen. In these tests the server was in the single-home mode (due to the single-home nature of TCP). The attack was directed to the same port used for the client-communication. The server and the client programs were adapted to use TCP; the Linux operating systems was chosen. To generate the TCP SYNs, I used the code of "juno.c" [28] at the attacker host. The TCP SYN packet as well as the SYN/ACK packet had a usual length of 26 byte. The procedure of the test was the same as described for the SCTP tests. Figure 19 shows the results of the tests: at the 50 Mbps link the TCP payload throughput is mostly between 30 and 35 Mbps during an 8 Mbps SYN flooding attack – with and without cookie mechanism. With the same configuration set the INIT attacks reduced the payload throughput down towards only 13 Mbps. The same can be seen in figure 20 with the 100 Mbps server-client connection: While the INIT attack lowered the payload throughput again to only 13 Mbps, the TCP server kept sending with a rate of about 45 Mbps – no matter if with or without cookie mechanism.

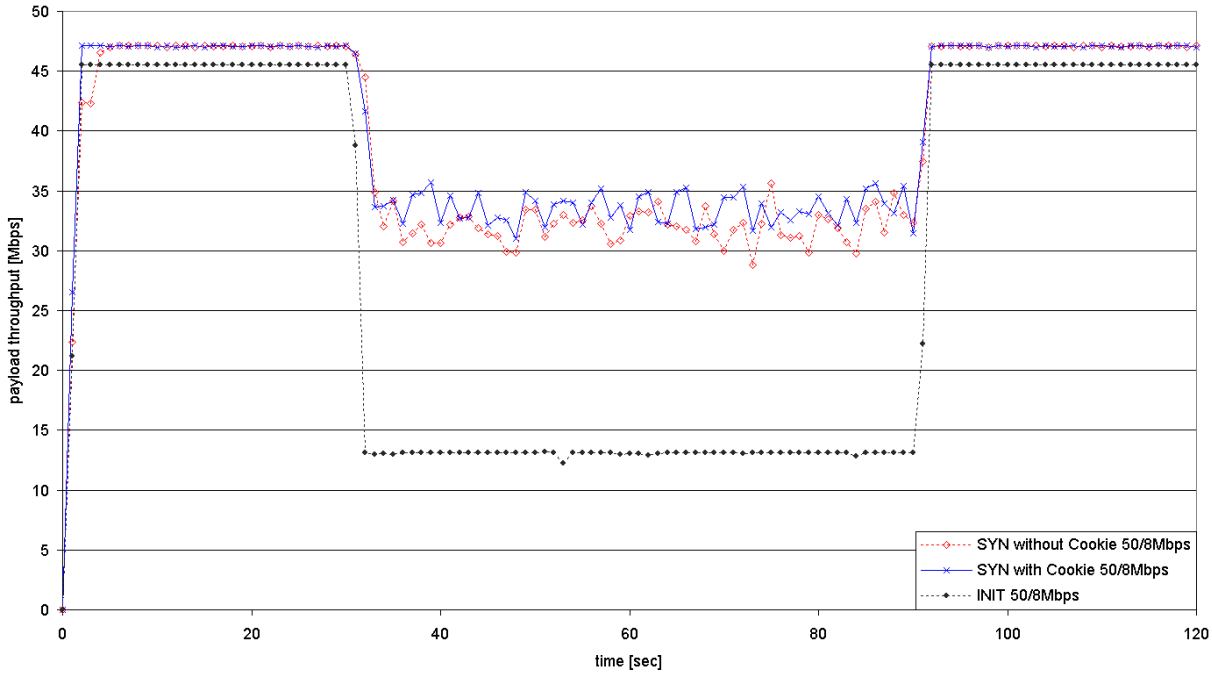

Figure 19: SYN flooding versus INIT flooding during 50 Mbps server-client transmission

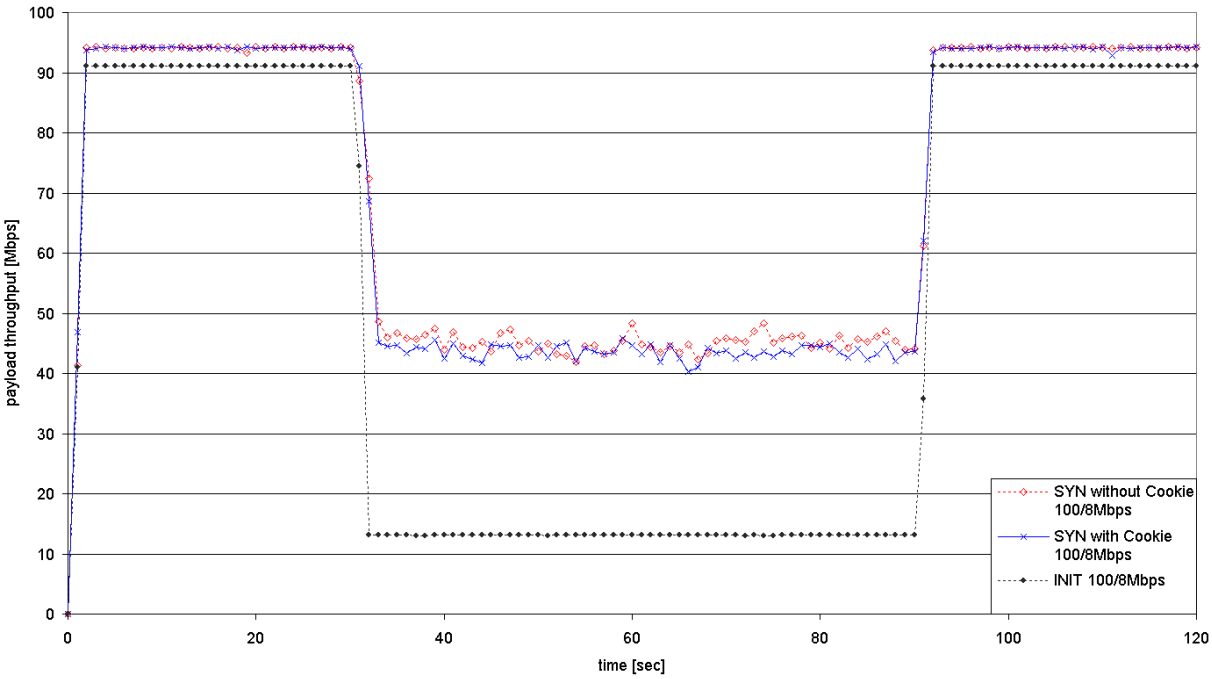

Figure 20: SYN flooding versus INIT flooding during 100 Mbps server-client transmission

3.4 Measurement conclusion

This part of the work dealt with the evaluation of SCTP's robustness against DoS attacks. For checking the robustness we chose measurable values. In chapter 3.3.1 we monitored the reliability of the server's response. We measured the rate of answered and unanswered INITs during a flooding attack and clocked the RTTs to evaluate the response delay of the answered INITs. We have seen that SCTP's loss rate went up roughly towards 50 % at a medium barrage size. The further increasing of the barrage size caused a reduction of the lost (i.e. unanswered) INIT rate. The sensibility of the server at a certain barrage size is caused by context swapping, the number of INITs per time slot is particular adverse. In case the barrage size is above this point, there are more INITs to answer at each time slot, which leads to relatively less context swaps and to load relieving of the CPU. At larger barrage sizes the loss rate remained strictly under 10 %. This can be considered as certain level of reliability and stability. By contrast the TCP peer without state cookie mechanism lost de facto all SYNs of the barrage which arrived later than the first 770 SYNs; i.e. about 98 % loss rate when the barrage has a feasible size of 35.000 packets. This effect is caused by the backlog limit. It remains exhausted during the barrage. Under attack this kind of endpoint can not be considered as reliable nor robust. The TCP endpoint *with* cookie mechanism delivered other results: the loss rate was almost zero at all barrage sizes. This marks a very performance-effective implementation of the cookie mechanism. But we have to keep in mind that every implementation is a trade-off between cryptographic security – that should avoid attacks against memory/buffer resources – and economy of CPU time – that avoids CPU resource attacks (see chapter 4.2.2.1 of [3] and also pp. 218 of [29]). The TCP state cookie protection might be cryptographically weaker. There were already exploits of former TCP implementations to generate valid cookies and subsequently to run sensitive buffer resource attacks. Another fact is that the TCP cookie can encode only a very limited amount of TCP options and parameters which leads to the problems already mentioned in chapter 3.2. This limited functionality enables a less complex computation and explains the performance advantages. The results of the RTT analysis confirm this: The TCP endpoint answers significantly faster than the SCTP endpoint. Nevertheless the RTTs during large INIT barrage attacks are also stable and do not increase any more beyond a certain barrage size where the context swapping effect has no significance any more.

In chapter 3.3.2 we focused on other values that are an alternative measurement for the robustness of SCTP against DoS attacks: the quality of service related to an existing server-client association was monitored by measuring the decline of the transmission rate of the victim during the attack. All imaginable kinds of flooding attacks for exploiting possible vulnerabilities of SCTP's handshake mechanism were performed in different configuration sets. One assumption was the limit of the bandwidth that was available for the attacker to stress the

victim. The outcomes are strength classifications of the different attacks and configurations. They can be summarised by inequations:

(1) INIT > LONG-INIT > REAL-COOKIE > RND-COOKIE

(2) primary interface > backup interface

(3) open port > closed port

Inequation (1) shows that the INIT flooding attack is the strongest kind of attack. An attacker with a given link bandwidth cannot gain more effect with using the other fake packets. There are no unexpected weaknesses of the SCTP implementation. The INIT flood causes the strongest CPU utilisation. Inequation (2) points the higher effect when flooding towards the same (i.e. primary) interface used for inoffensive data transmission. Inequation (3) shows the higher vulnerability if the attack is directed to an open port than to a closed port.

To mitigate the effect of attacks it might be possible to force the more robust configurations during an attack. The kind of attack (seen in inequation (1)) can exclusively be chosen by the attacker. But the other configurations can possibly be influenced by the victim: an SCTP server could be run at an unusual port to confuse the attacker and to keep the open port secret. Additionally several interfaces could deflect from the primary interface used for inoffensive and established associations. However, these approaches will not completely solve the limited but remaining vulnerabilities.

The last presented measurements of 3.3.2 allow a comparison with TCP. Also here, TCP shows a higher robustness; the data transfer is significantly less influenced by the SYN attack. The explanation was already noted above: Generating the INIT/ACK packet with its more complex cookie consumes more computing resources than generating the SYN/ACK packet with the 'poor' cookie. During the higher utilisation of the SCTP system, data of the established association cannot be sent as fast as desired.

An additional explanation of the higher performance of TCP can be the fact that the TCP implementation was developed and improved over decades of years. The code is very optimised; ineffective and disadvantageous parts were substituted in the course of time. SCTP may functionally also be well-developed; nevertheless it is still in a relatively early phase of development and the implementations are surely improvable concerning the performance.

4 Performance of security solutions

4.1 Motivation and introduction

Standard SCTP does not provide security features like authentication or confidentiality. To increase the acceptance of SCTP as a multi-purpose transport protocol, the lack of security needs to be eliminated: additional functionality has to be provided to secure SCTP associations including its payload.

There are already possibilities to provide authentication and confidentiality for SCTP. IPsec as a secure network protocol can be combined with SCTP. Alternatively TLS can be used to work on top of SCTP (according to the protocol layer model). Both combinations of standard security protocols with SCTP lead to functional and performance related problems. To solve these problems Secure-SCTP (S-SCTP) was developed; it is assumed to be optimal because the security functions are directly integrated into SCTP.

These three solutions can be compared by focussing on qualitative and quantitative criteria. While security features, flexibility and ease of usage, compatibility and other functional features can be compared in a qualitative way, performance is a very important quantitative criteria for the acceptance of a solution and needs to be compared.

Additionally, quantitative tests may allow conclusions to qualitative statements: if an implementation is still working properly during high load, the robustness is high and in case of DoS attacks it is more invulnerable and therefore more reliable and secure.

In chapter 4.2 the theoretical background is described shortly, the features of the different solutions are compared and especially the limitations are pointed out. Chapter 4.3 presents the performance evaluation of these solutions. The comparison of the results leads to the conclusion in chapter 4.4.

4.2 Theoretical Background

In chapter 2.2 we already presented the security solutions for SCTP. Because IPsec and TLS were not developed for SCTP, these security solutions have limitations and problems. Table 2 provides an overview of all three security solutions. They are theoretically compared by evaluating how good they support different features. These features are in each case well supported (indicated with “+”), not very well supported (shown by a “–“) or not supported at all (signified by “no”).

Criteria	*TLS*	*IPsec*	*S-SCTP*
Scalability for multiple streams	–	+	+
Support for SCTP multi-homing	+	(–)	+
Overhead for small messages (bundling)	–	+	+
Overhead for long messages (fragmentation)	+	–	–
Protection for unordered delivery service	no	+	+
Protection for SCTP control chunks	no	+	+
Flexible multiplexing of secure/insecure streams	+	no	+
Management of security sessions (handling, automation)	+	–	+
Partial Reliable Transport (SCTP extension)	no	+	+
Dynamic Address Reconfiguration (SCTP extension)	+	–	+

Table 2: Comparison of security solutions [30]

There are *functional* limitations due to new features introduced by SCTP, which cannot be supported by the standard security solutions. These limitations can be found in table 2 where a "no" is written. These features are not supported by the respective protocol set:

- The *unordered delivery service* of SCTP cannot be used with TLS over SCTP. TLS was developed for TCP and uses the fact that all packages are delivered reliable and in sequence.
- Caused by the same reason, the proposed *partial reliable delivery* extension cannot be supported by TLS over SCTP. For TLS – as an extension of TCP – strict reliability is mandatory.
- A functional security problem of TLS over SCTP is the lack of *protection of the SCTP control chunks*. Because TLS only protects the user data – which takes place at a higher layer than the SCTP layer, protected content is just injected into the data chunks. All chunk headers and control chunks and the common header are not protected at all.
- With SCTP over IPsec *flexible multiplexing of secure and insecure data* is not possible, because the whole SCTP traffic is encapsulated by IPsec. Separating secure from insecure streams is not possible.

Other problems are *performance-related*. They are marked with a "–" in table 2:

- *Scalability of multiple streams* is limited for TLS over SCTP. For each stream a new TLS session initiation has to be performed as well as rekeying during already established TLS sessions. This can be a performance problem in case there is a large number of streams.
- Another problem is the *overhead for small messages*, because each message is encrypted separately by TLS. The other solutions avoid this effect by bundling small messages into

one SCTP packet which is encrypted as a whole. Thus TLS produces more header data added to each encrypted block. This header data uses bandwidth, too. TLS also causes a higher frequency of cryptographic function calls which leads to more CPU workload.

- The *multi-homing* feature is not well supported by SCTP over IPsec. Security Association (SA) management and key management of IPsec were developed to cope with unique IP addresses for each peer. In general key and association management is complex. An SCTP association between multi-homed endpoints with X and Y numbers of interfaces has up to X * Y different paths. This would lead to 2 * X * Y numbers of IPsec associations – due to the one-way nature of IPsec associations – which need to be managed separately. Although there is a new standard allowing SAs with more than one IP address per peer, there are no adequate implementations available until now (see [14], [17]).
- The same problem occurs with the *dynamic address reconfiguration* extension in combination with IPsec. As an extension of the multi-homing feature it would increase the complexity of managing security associations of IPsec. New IPsec associations would need to be established and old ones would need to be terminated even during a single and permanent SCTP association.
- IPsec also has a problem with *long messages* which are longer than the PMTU and hence must be fragmented. This causes additional overhead because each fragment has to be protected separately.
- As mentioned before S-SCTP has similar problems with the fragmentation of long messages. Fragmented S-SCTP payload needs to be encrypted separately. This fact also causes overhead and can reduce the performance.

The first mentioned *functional* problems of the different solutions cannot be solved. At least they cannot be solved without essential changes of these protocols. But *performance-related* problems mentioned above are only identified theoretically up to now. To evaluate the actual significance of these problems the solutions have to be analysed and real performance tests have to be made. The purpose of the following scenarios is to quantify these performance-related effects of the different security solutions.

4.3 Measurements

To evaluate the performance of these protocol sets in a realistic way, we need a realistic environment. Like mentioned in chapter 3, we need an environment, where we control all parameters regarding the hosts and the network to be able to compare the performance of the different security solutions. This is why we cannot use real networks like the Internet, instead we use the self-implemented testbed presented in my project seminar work (see [24]). The same

and unchanged testbed configuration was already used for the measurements in chapter 3.3.1. In the following sub-chapters three different kinds of tests are treated.
SuSE Linux 9.3 with kernel 2.6 is installed at the endpoint hosts. As SCTP implementation *sctplib* [31] was used. This userland prototype implementation has its advantages for research work. It is easier to install bug-fixes and other updates; there is no need to touch the kernel of the operation system. Adaptations can be made easier and the programs could be analysed with less effort. Furthermore the Computer Networking Group has experience with it and knowledge about it. The most important reason to use sctplib is that the implementations of TLS over SCTP and S-SCTP – which should be evaluated – are extensions of sctplib and cannot be run with e.g. the kernel implementation of FreeBSD. These already available sctplib extensions are introduced in [14]. Their implementations include basic server-client-programs to perform data generation, transmission and analysis. Furthermore the IPsec programs of SuSE (ipsec-tools) were used. All details about setting up the sctp userland implementation, their security solutions and the usage of the test programs are described in my earlier project report [24]. The complete results of the following tests are also at the compact disc.

4.3.1 Comparing throughput with different segment size

The following tests analyse the payload throughput of the different security solutions. Like in a real environment the bandwidth of the link is limited. Dummynet simulates different links: first a DSL (digital subscriber line) uplink connection with 192 kbps (kilobit per second) is imitated, further tests are also done with the link bandwidths of a T1 (Trunk 1) link (i.e. 1.544 kbps) and – without dummynet – the actual link bandwidth of Gigabit Ethernet (1.000 Mbps). All specifications are meant full-duplex. The test programs (“test-ssctp-serv.cpp”, “test-ssctp-cli.cpp”, “test-mssl-serv”, “test-mssl-cli.cpp”) are used to run the data transmission between server and client. To get comparable configurations of all three security solutions, the same cryptographic standards need to be chosen. All three solutions support triple DES (triple data encryption standard, 3DES) as data encryption standard to gain data confidentiality and SHA (secure hash algorithm) as cryptographic hash function to ensure data integrity, so these algorithms are used. As measurement reference the unchanged sctplib without extra security functionality is tested, too.

To gain results about the different security solutions' behaviour, segments with different sizes are sent to the peer. Especially the effect of different overhead at small and large segment sizes should be evaluated. During these tests server and client are multi-homed with two interfaces each. The data is sent via a single stream. Every single test is performed in this way: the server program starts at the server host, then the client starts at the peer. After the 4-way-handshake the server begins sending the data as fast as possible, only limited by the flow

and congestion control. The client reads out the buffer without delay, calculates the throughput in intervals of one second and after five minutes it issues the mean receive rate. All tests are performed five times.

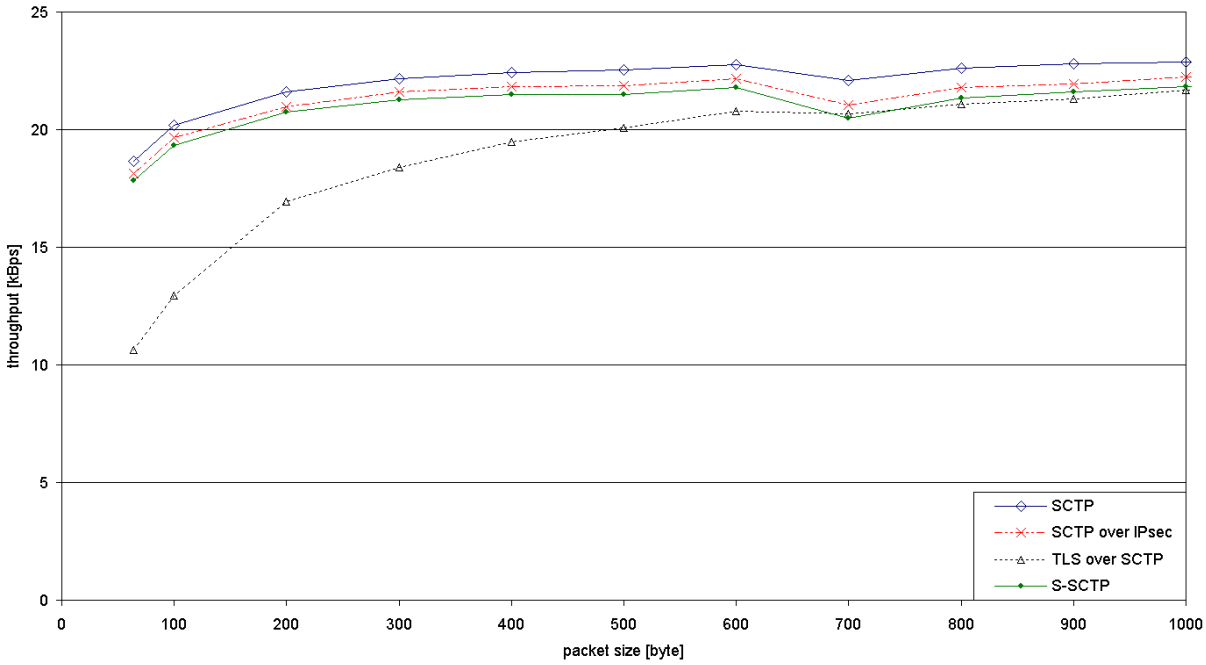

Figure 21: Payload throughput of security solutions at DSL (192 kbps) link simulation

Figure 21 shows the results of the DSL simulation. Each curve represents the mean of the particular measurements. The maximum bandwidth of this link is exactly 24 kBps. The top-most curve shows the standard SCTP results. It starts at 64 byte segment size with a value of 18.7 kilobytes per second (kBps). Higher segment sizes cause higher payload throughput. A maximum of about 22.7 kBps payload throughput is reached at a segment size of 600 bytes. At this segment size the overhead caused by the chunk headers is relatively low compared to the smaller segment sizes. Additionally the overhead caused by the SCTP common header is reduced with the help of bundling: more than one data chunk is sent in a single SCTP packet. The link bandwidth is used completely, during the test the CPU utilisation is significantly under 100%; therefore we can state that the so-called bottleneck is the link bandwidth limitation. At the segment size of 700 bytes is a slight dent in the curve (22.1 kBps). If the segments have 700 bytes or more, bundling is not possible any more due to the PMTU. The overhead is again higher although the chunk header forms a smaller portion of the complete chunk.

The curves of SCTP over IPsec and S-SCTP have the same shapes, but they are located slightly (i.e. less than 1 kBps) under the SCTP curve. Bundling is performed analogue to the

standard SCTP transmissions. The only difference is the higher overhead: IPsec has additional ESP headers and authentication blocks; S-SCTP submits additionally its 'EncData' header and the AUTH chunk. Both protocols add this additional data once per SCTP packet or IP packet respectively. TLS over SCTP is different at this point: each single user message gets its own record header and authentication part before it is inserted into an SCTP data chunk. If the data chunks are small enough, SCTP bundles several of them into one SCTP packet – including as much record headers and authentication blocks as user messages. This disadvantage can be seen at the TLS over SCTP curve of figure 21: the payload throughput for small user messages is very low (10.6 kBps at 64-byte messages) compared to the other security solutions. Further on the curve converges to the other ones and meets them at the segment size of 700 bytes. At this segment size there is no bundling effect any more and one SCTP packet (or one IP packet respectively) contains only a single set of authentication part and encryption header.

In another set of measurements the solutions are compared with a different link connecting the endpoints: a T1 link with a bandwidth of 1544 Mbps is simulated. All other parameters are exactly the same. The results can be seen in figure 22. The curves have exactly the same shapes and are exactly in the same positions related to each other. The chart looks like a copy of the already analysed one. The only difference is the scale of the y-axis. T1 allows a bandwidth of 193 megabyte per second (MBps), the payload throughput reaches 182 MBps at 600-byte segments. The CPU is still not under full utilisation; the bottleneck again is the link limitation. The differences of the security solutions have the same proportions compared to the DSL measurements. That is why also the number of IP datagrams, SCTP packets and security headers is proportionally higher than during the DSL tests. All the above mentioned explanations for the results are also valid for the results of this chart.

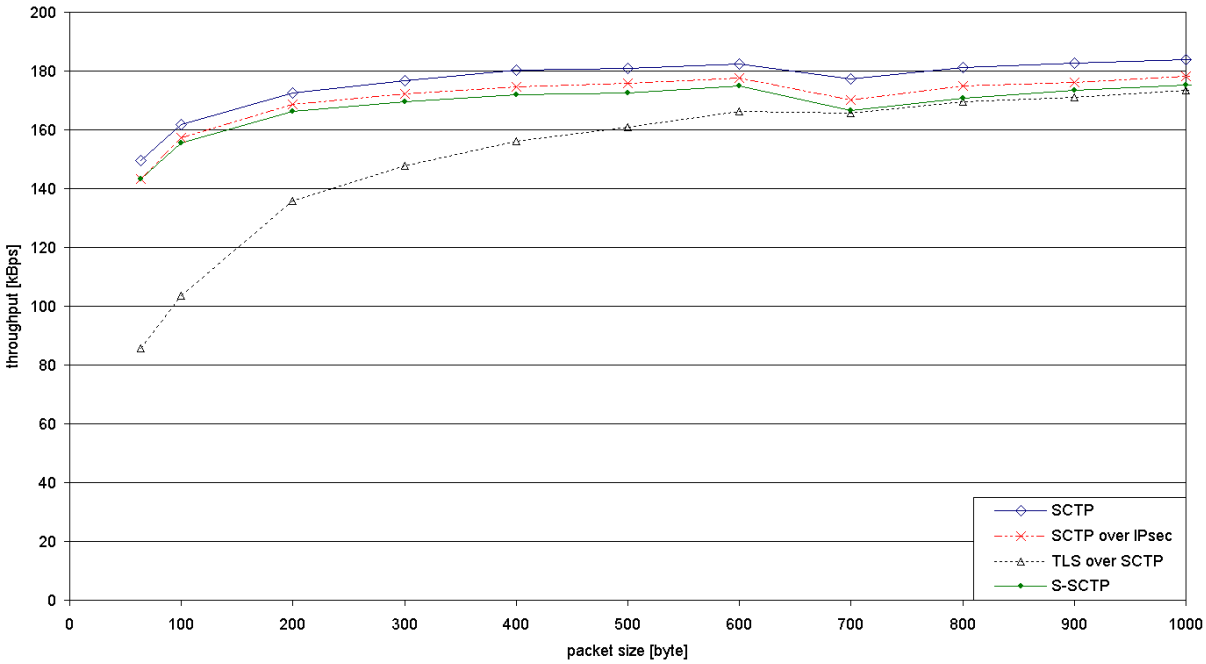

Figure 22: Payload throughput of security solutions at T1 link simulation

A third set of measurements is performed with exactly the same configuration apart from the link's bandwidth. This time it was not limited by dummynet and therefore the full 1000 Mbps of the Gigabit Ethernet are available. CPU monitoring has shown a complete utilisation during the tests for all segment sizes and security solutions. The available bandwidth is not completely used at all. In these cases the link limitation is not the bottleneck. In fact the protocol, its implementation and the CPU performance limit the actual payload throughput. Therefore these tests do not allow statements about the traffic overhead and the resulting network effectiveness. Figure 23 shows the results. The top-most curve describes the standard SCTP tests, between the segment sizes of 64 bytes and 1300 bytes there is a steady and fast increasing of the throughput – starting at 1129 MBps and resulting in 10,340 MBps. Unlike the curves seen before there is no significant dent at 700-byte segments. It does not affect the CPU utilisation much if the unencrypted data chunks are bundled or not. Obviously there is a meaningful offset between 1300-byte and 1400-byte segments; here is the threshold between no fragmentation size and fragmentation size. At 1400-byte segments the server has to send about double as many SCTP packets in the same time to reach the same payload throughput compared to the transmission of 1300-byte segments. This is much more CPU time-consuming.

The curves of SCTP over IPsec and S-SCTP remain much underneath the standard SCTP curve. This is not amazing because cryptographic functions utilise the CPU; encryption of a data block and creating an authentication part has to be done – once each SCTP packet or IP packet respectively. Offsets can be seen between 600-byte and 700-byte segments. It is the bundling/non-bundling threshold. At 700-byte segments about twice as many packets need to be sent. For each packet the encryption and authentication functions must be called. So the cryptographic workload of the CPU is much higher; this is why the throughput cannot reach the same rate as before. Further on the throughput increases again due to the larger blocks that are encrypted. At 1400-byte segment size there is the fragmentation threshold. Like mentioned before the average encrypted block size is low, the cryptographic functions need to be called relatively often. Further on the payload throughput increases because the average size of the fragmented parts increases, too. This affects IPsec more heavily than S-SCTP.

The TLS curve is somewhere above the other security solution curves. Especially at the bundling and fragmentation thresholds the offsets are minimal and the curve remains at a significant higher level. This seems to be curious because S-SCTP uses the same cryptographic libraries of TLS. S-SCTP and TLS over SCTP should deliver the same results between 700-byte and 1400-byte segments. In this range the cryptographic functions are called with the same frequency. Actually, S-SCTP and SCTP over IPsec are slower because the encryption of the data and the transmission take place in the same process. When the send queue of this process is full, the send call blocks and waits until new packets can be transmitted. During this time the process runs idly. In the case that bundling cannot be used any more (around 700 bytes of user data length) or when long messages have to be fragmented (1400 bytes of user data length) the throughput drops because there is less user data contained in packets and the process cannot send more packets due to the blocking send call. When using TLS, encryption and transmission are performed by different processes. In this case, even if the send call blocks and waits until new packets can be transmitted, TLS can still encrypt data for future transmission.

Finally an additional statement can be gained out of figure 23: TLS and S-SCTP have the ability to multiplex protected and unprotected data. The unprotected data throughput of standard SCTP was meaningful faster; a mixture of protected and unprotected data would consequently lead to a performance that is higher than at completely protected transmissions.

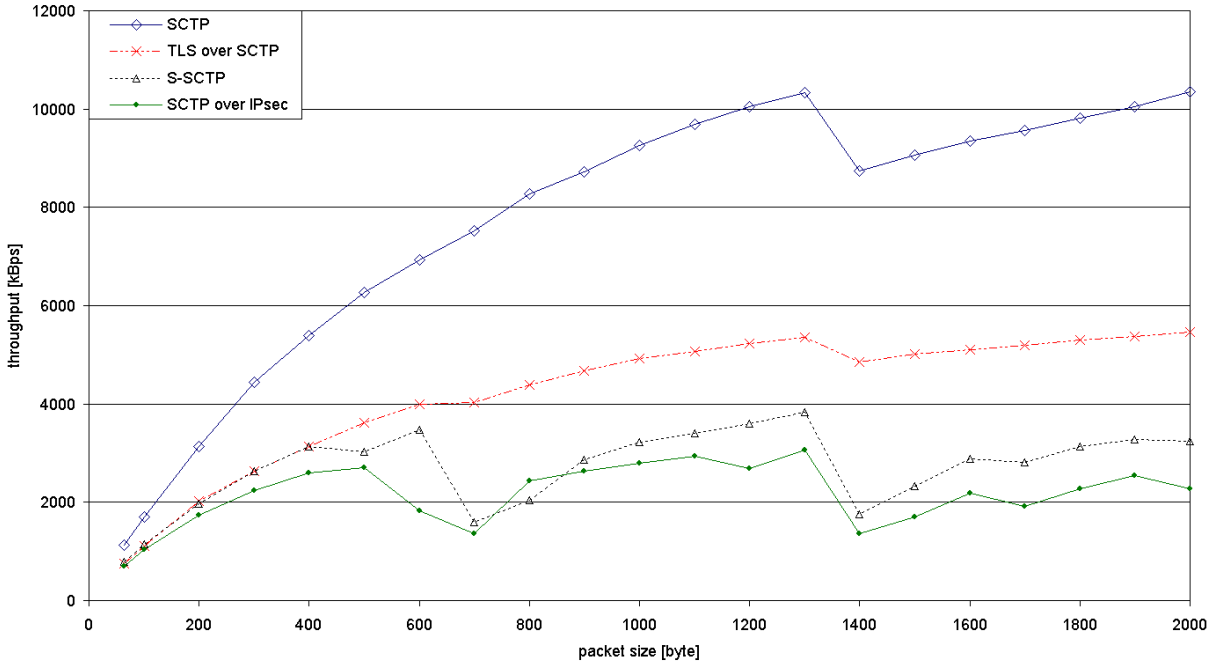

Figure 23: Payload throughput of security solutions at Gigabit Ethernet link

4.3.2 TLS handshake costs concerning the amount of streams

As already pointed out in chapter 4.3 the scalability for multiple streams is theoretically a problem with TLS over SCTP due to the separate security association management for each stream. Especially at the beginning of these associations the handshakes have to be performed. To evaluate the actual significance of this drawback measurements are performed. Different amount of streams for the maximum inbound streams (MIB) and outbound streams (OS) are chosen. The delay between the start of the client program execution and the transfer of the first DATA chunk is measured. The results are depicted in figure 24. Obviously the duration of setting up the security associations is proportional to the amount of streams. The bandwidth of the link effects the gradient of the curves. While looking at the absolute values, the significance can be evaluated as very high. When there are 1000 streams to initiate, it takes 8 seconds at Gigabit Ethernet links, 16 seconds at T1 links and 69 seconds at DSL-like links. At 10,000 streams it takes 97, 175 and 704 seconds (i.e. more than 11 minutes!) respectively. During this time CPU and network are well utilised.

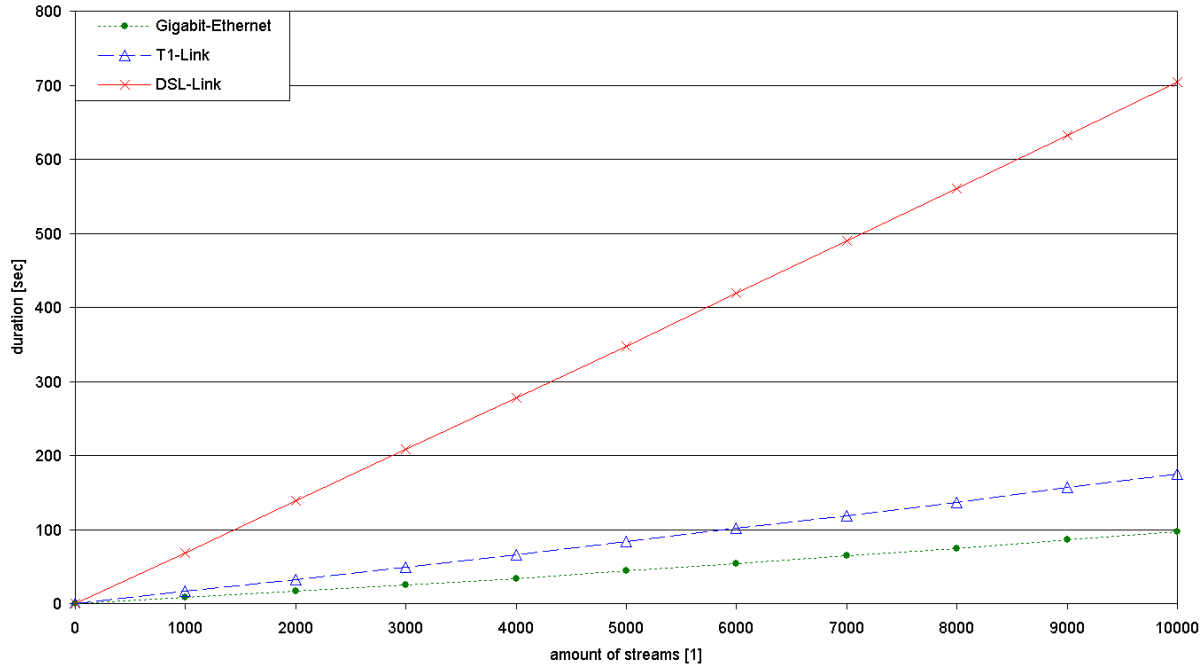

Figure 24: Setup time for security associations of TLS over SCTP

4.3.3 Throughput with an erroneous link

Now we want to focus on the behaviour of the security solutions regarding erroneous links. The aim is to simulate low quality of the data link or network layer. Due to the possible usage of SCTP with its security solutions as a general purpose transport protocol of modern networks, the underlying link could be of wireless nature. Typical for wireless networks are limited bandwidth and also limited quality of service. High delay, jitter and loss are the problems the reliable transport protocol has to face. SCTP has to compensate the lack of reliability and needs to provide reliable transport towards the upper layers. The error correction mechanisms of standard SCTP are known. Here, real measurements should show the effectiveness of these mechanisms for standard SCTP and all three security solutions.

The loss of packages can be considered as the worst disturbance of low-quality networks. These tests are done while simulating a link with limited bandwidth and adjustable loss rate. Dummynet is able to drop IP packets randomly with a certain probability and the resulting occurrence. As bandwidth limitation the T1 example of 1,544 Mbps is chosen again. The loss rate is changed between 0% and 15%. Each test lasts 5 minutes. Segments of 1000 bytes are sent from the server to the client. The results are displayed in figure 25. The standard SCTP payload throughput is about 180 MBps when the loss rate is 0%. The curve goes steadily

down without any special dent or bend. At 15% of loss only a throughput of 40 MBps remains. Relative to the high loss rate this remaining throughput can still be considered as satisfying. But more remarkable is that all security solutions show the same behaviour. The curves are not significantly different. At limited links all three security solutions and standard SCTP show the same robustness and performance at the certain loss rates.

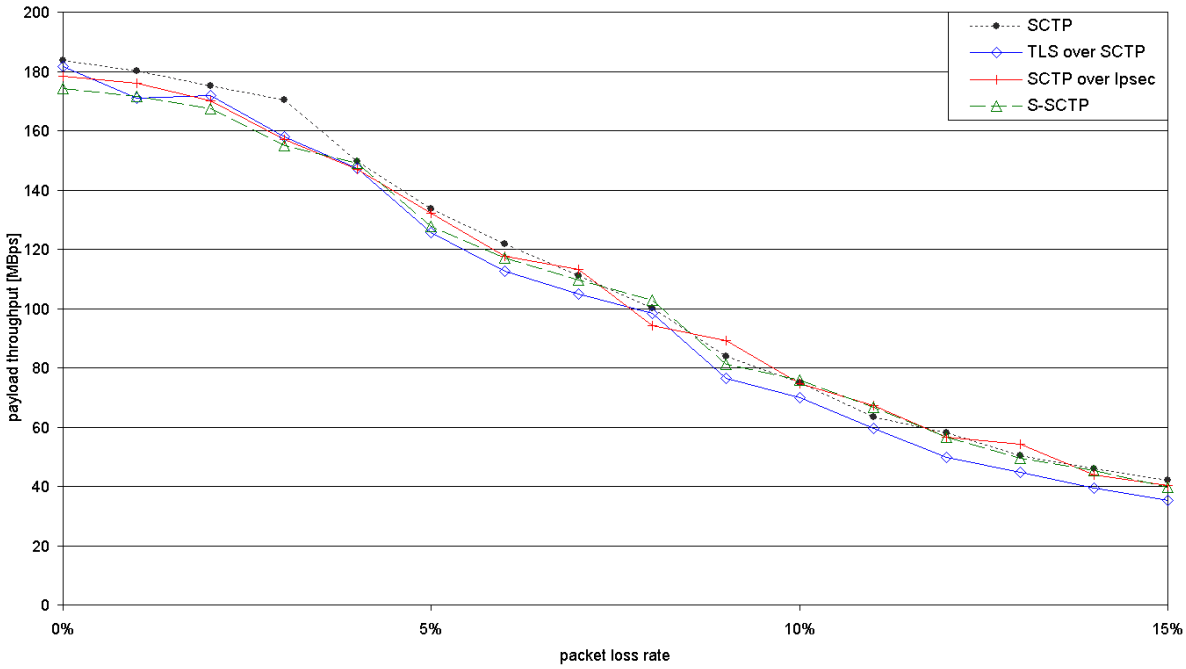

Figure 25: Payload throughput at different loss rates

4.4 Measurement conclusion

In this part of the work we focused on features and limitations of the SCTP security solutions. Functional drawbacks of the alternatives 'SCTP over IPsec' and 'TLS over SCTP' were pointed out. The new solution S-SCTP was introduced to solve these functional disadvantage. The performance-related problems were discussed theoretically in chapter 4.2, too. To obtain statements about the significance of these performance-related problems, measurements have been performed. Let us have a look at the treated problems:

- *Scalability of multiple streams:* This limitation is an essential drawback for TLS over SCTP. If there is a high amount of streams that need to be encrypted – which is imaginable for the use in telecommunication signalling networks –, the establishment of new security associations are very time-consuming and even unacceptable. Potential rekeying scenarios

are also affected although their handshakes are shorter and the impact is not as high as at the initial handshakes. As expected the other solutions have shown their good scalability.

- *Overhead for small messages:* This drawback was confirmed by the measurements with the limited link bandwidth as a bottleneck. But the performance drawback in fast networks – which depends mostly on the CPU utilisation – was less significant than expected.
- *Long messages:* IPsec was the weakest solution facing the fragmentation problem. But also S-SCTP was affected by throughput reduction.

Over high bandwidth links, the throughput of IPsec is lower compared to the other solutions. If only a small portion of the transmitted data has to be secured, the throughput of IPsec is lower because it can not differentiate between data that has to be secured and data that can be send unsecured. A TLS over SCTPs major performance limitation is linked to the number of streams which have to be secured. With an increasing number of streams, the memory usage and the time to establish the secure sessions for all streams also increase. The only identified throughput degradation occurs when TLS has to secure small messages which can be bundled. S-SCTP was designed to overcome the performance limitations of SCTP over IPsec and TLS over IPsec, so we only identified some performance limitations using high bandwidth links. The main reason for this is the use of a prototype S-SCTP implementation developed to validate the design decisions which was not optimised for performance yet.

5 Conclusion

This work deals with the analysis of robustness and security of SCTP. Chapter 1 outlines briefly the importance of robustness and security, presents the motivation of analysing SCTP concerning these aspects and introduces the general approach of realisation.

In chapter 2 a short overview of SCTP is given. Further on the most important differences between SCTP as a new, and TCP as a traditional transport protocol are pointed out. Essential attributes of SCTP – like multi-homing, message-wise transportation and multi-streaming – are the basis for a set of features that is different to the TCP's one. In the third sub-chapter the standard security protocols IPsec and TLS are explained; furthermore the combinations 'SCTP over IPsec' and 'TLS over SCTP' as SCTP security solutions are explained. Additionally Secure-SCTP (S-SCTP) is introduced; this integrative approach was developed to solve the functional limitations resulting from the other solutions.

Chapter 3 covers the analysis of robustness. Vulnerabilities of transport protocols towards DoS attacks and the motivation to mitigate their effects are described. DoS attacks are analysed in a theoretical manner and countermeasures are derived. Actually used techniques of the traditional transport protocol TCP and of the new SCTP are displayed. Further on tests were developed to evaluate the actual efficiency of these countermeasures. Different kinds of attacks, different configurations regarding the testbed and different parameters to measure the robustness are depicted.

Finally the robustness of standard SCTP has shown good results concerning memory/buffer resource DoS attacks. A consequently implemented cookie mechanism has actually not shown any vulnerabilities to allow a memory/buffer-exhausting exploit. More differentiated results were obtained by evaluating the CPU resource attacks. The INIT attack towards an SCTP implementation shows higher vulnerability than a SYN attack towards a current, state cookie-protected TCP implementation. One reasons is the more complex state cookie of SCTP. Additionally a comprehensive set of other possible attacks were tested, too. Unexpected or even higher vulnerabilities were not discovered. Approaches of mitigating the impact of DoS attacks have been shown: hiding the information about open ports and primary paths to potential attackers reduces the effects of the attack. Finally and as mentioned before SCTP is still in a early phase of development and the implementations are surely improvable concerning the performance. It is imaginable that future implementations of SCTP are less sensitive to INIT attacks than current TCP implementations to SYN attacks.

Chapter 4 deals with the security solutions for SCTP. Standard SCTP has no security features to protect the transmitted data sufficiently for today's fields of application. Three already introduced security solutions are compared. In a deductive way their problems are discovered. Some of these problems are functional, especially some features of SCTP cannot

be used in the combination with the standard security protocols. S-SCTP solves these problems and has less restrictions due to its integrated implementation. Other non-functional problems are performance-related drawbacks. In the first step they were also deducted theoretically. In a further step their significance is evaluated by measuring the actual performance of the different solutions in different scenarios.

IPsec over SCTP has shown the most disadvantageous results. Beside a lack of flexibility, scalability and controllability, it shows the most performance-related problems. TCP over SCTP also has essential functional limitations, but the performance-related drawbacks were actually not very serious apart from the scalability problem with a high amount of streams.

Basically the concept of S-SCTP has solved the problems occurring at the other solutions with traditional security protocols. The performance is not as good as TLS over SCTP; this is caused by the lack of performance optimisation of the used prototype implementation.

What has not been discussed yet is the dependency of S-SCTP's acceptance in the standardisation process. The IETF standardisation process of S-SCTP would take a long time and requires essential changes of the existing SCTP standard. Without an adopted standard there is almost no chance that this solution will be accepted widely. Especially kernel implementations for all important operating systems are necessary to increase the acceptance. But problems would for example occur when the S-SCTP's association and key management is implemented into the kernel. This could cause unpredictable blocking effects due to certificate verification and key establishment procedures. This could effect the responsiveness of the operating system and thus would not be accepted by developers and users. Aware of these problems the Computer Networking Group is developing a new concept: *Datagram TLS (DTLS) aware SCTP*. This solution uses the standard of SCTP without major modifications in combination with DTLS; a modified version of TLS to support UDP as the underlying transport protocol. It solves the earlier shown problems of TLS over SCTP and provides functionality similar to S-SCTP. The concept is presented in [30]. The current work of the group is to refine the concept. An Internet Draft is already submitted to the IETF [32]. Further on a prototype implementation will be developed to advance a security solution for SCTP with a wide acceptance.

References

[1] Stewart, R.; Xie, Q.; Morneault, K.: RFC 2960 – "Stream Control Transmission Protocol", IETF, Network Working Group, October 2000

[2] ISO 7498:1984 Open Systems Interconnection - Basic Reference Model

[3] Stewart, R.; Xie, Q.: "Stream Control Transmission Protocol – A Reference Guide", Addison-Wesley, November 2001

[4] Stone, J.; Stewart, R.; Otis, D.: RFC 3309 – "Stream Control Transmission Protocol (SCTP) Checksum Change", IETF, Network Working Group September 2002

[5] Stewart, R.; Ramalho, M.; Xie, Q.; Tüxen, M.; Conrad, P.: RFC 3758 – "Stream Control Transmission Protocol (SCTP) Partial Reliability Extension", IETF, Network Working Group, May 2004

[6] Stewart, R.; Ramalho, M.; Xie, Q.; Tüxen, M.; Conrad, P: Internet-Draft – "Stream Control Transmission Protocol (SCTP) Dynamic Address Reconfiguration", draft-ietf-tsvwg-addip-sctp-14 (work in progress), IETF, Network Working Group, March 2006

[7] Tüxen, M.; Stewart, R.; Lei, P.; Rescorla, E.: Internet-Draft – "Authenticated Chunks for Stream Control Transmission Protocol (SCTP)", draft-ietf-tsvwg-sctp-auth-02 (work in progress), IETF, Network Working Group, March 2006

[8] Balliache, L.: "Practical QOS", http://www.opalsoft.net/qos/TCP-1021.htm

[9] Floyd, S.: RFC 2914 – "Congestion Control Principles", IETF, Network Working Group, September 2000

[10] Jungmaier, A.; Schopp, M.; Tüxen, M.: "Performance Evaluation of the Simple Control Transmission Protocol (SCTP)", ATM 2000 – Proceedings of the IEEE Conference on High Performance Switching and Routing, 2000, pp 141-148

[11] Ong, L.; Yoakum, J.: RFC 3286 – "An Introduction to the Stream Control Transmission Protocol (SCTP)", IETF, Network Working Group, May 2002

[12] Coene, L.: RFC 3257 - "Stream Control Transmission Protocol Applicability Statement", IETF, Network Working Group, April 2002

[13] Jungmaier, A.: "SCTP for Beginners", http://tdrwww.exp-math.uni-essen.de/inhalt/forschung/sctp_fb, 2003

[14] Unurkhaan, E.: "Secure End-to-End Transport – A new security extension for SCTP", Dissertation, March 2005

[15] Mathis, M.; Mahdavi, J.; Floyd, S.; Romanow, A: RFC 2018 - "TCP Selective Acknowledgement Options", IETF, Network Working Group, October 1996

[16] Baccala, B.: Connected: "An Internet Encyclopedia", http://www.freesoft.org/CIE/

[17] Bellovin, S.; Ioannidis, J.; Keromytis, A.; Stewart, R.: RFC 3554 – “On the Use of Stream Control Transmission Protocol (SCTP) with IPsec”, IETF, Network Working Group, July 2003

[18] Jungmaier, A.; Rescorla, E.; Tüxen, M.: RFC 3436 – “Transport Layer Security over Stream Control Transmission Protocol”, IETF, Network Working Group, December 2000

[19] Unurkhaan, E.; Rathgeb, E.; Jungmaier, A.: “Secure-SCTP – A versatile and secure transport protocol”

[20] daemon9, "Project Neptune", Phrack Magazine, Volume 7, Issue 48, File 13 of 18, July 1996.

[21] Eddy, W: Internet-Draft – “TCP SYN Flooding Attacks and Common Mitigations”, draft-eddy-syn-flood-02 (work in progress), IETF, Network Working Group, April 2006

[22] Aura, T.; Nikander, P.: “Stateless Connections”, Proc. Information and Communications Security, First Intern. Conf., ICICS’97, pp. 87–97, November 1997, Beijing, China

[23] Bernstein, D.J.: “SYN cookies”, http://cr.yp.to/syncookies.html

[24] Nordhoff, M.: “Design and implementation of a test scenario to evaluate end-to-end security solutions for SCTP”, Project Seminar Report, IEM, May 2006

[25] Lemon, J.: "Resisting SYN Flood DoS Attacks with a SYN Cache", BSDCON 2002, February 2002

[26] Rizzo, L.: ”Dummynet”, http://info.iet.unipi.it/~luigi/ip_dummynet

[27] Tüxen, M.: “The SCTP testtool (STT)”, http://www.sctp.de/sctp-download.html

[28] 'Sorcerer of DALnet': “Juno”, http://packetstormsecurity.nl/DoS/juno.c

[29] Benecke, C.: “Überlebensfähige Sicherheitskomponenten für Hochgeschwindigkeitsnetze”, August 2002, http://www.cert.dfn.de/team/benecke/00_dis.pdf

[30] Hohendorf, C.; Rathgeb, P.; Unurkhaan, E.; Tüxen, M.: “Secure End-to-End Transport Over SCTP”, ETRICS 2006

[31] Tüxen, M.; Jungmaier, A.: “sctplib”, http://www.sctp.de/sctp-download.html

[32] Tüxen, M.; Hohendorf, C.; Rescorla, E.: Internet-Draft – “Datagram Transport Layer Security for Stream Control Transmission Protocol”, draft-tuexen-dtls-for-sctp-00.txt (work in progress), IETF, Network Working Group, February 2006, http://bgp.potaroo.net/ietf/all-ids/draft-tuexen-dtls-for-sctp-00.txt

List of Abbreviations

CRC	Cyclic Redundancy Check
DSL	Digital Subscriber Line
DoS	Denial of Service
DTLS	Datagram Transport Layer Security
HTTP	HyperText Transfer Protocol
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IETF	Internet Engineering Task Force
IEM	Institute for Experimental Mathematics
IP	Internet Protocol
ISAKMP	Internet Security Association and Key Management Protocol
ISN	Initial Sequence Number
kbps	kilobit per second
kBps	kilobyte per second
MAC	Message Authentication Code
Mbps	Megabit per second
MBps	Megabyte per second
MIS	Maximum Inbound Streams
MSS	Maximum Segment Size
MTU	Maximum Transfer Unit
NIC	Network Interface Card
OOTB	Out Of The Blue (Packet)
OS	Outbound Streams
OSI	Open Systems Interconnection
PMTU	Path MTU
RFC	Request For Comment
RTO	Retransmission TimeOut
RTT	Round Trip Time
SHA	Secure Hash Algorithm
SI	Stream Identifier

SIGTRAN	Signalling Transport
SSL	Secure Socket Layer
SSN	Stream Sequence Number
T1	Trunk 1 (telecommunication standard)
TCB	Transmission Control Block
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TLV	Type-Length-Value (parameter)
TSN	Transmission Sequence Number
UDP	User Datagram Protocol
ULP	Upper Layer Protocol

Appendix

Hardware Configuration

Hardware of Server:

- AMD Athlon XP 2000+
- 2 * Netgear AC9100 Gigbit Ethernet
- 1* Broadcom 100Base-T onboard
- Asustek A7V8X Motherboard
- 512 MB Ram

Hardware of client and attacker:

- AMD Athlon XP 2000+
- 2 * Intel Pro/1000 MT Server Adapter
- 1* Broadcom 100Base-T onboard
- Asustek A7V8X Motherboard
- 512 MB Ram

Hardware of Router:

- Motherboard A8N-SLI
- AMD Athlon64 3000 Venice
- 1 GB RAM
- 2 * Intel Dual Server Adapter 1000Base-TX PCI64 (PWLA-8492MT)
- 1 * Marvell Gigabit Ethernet

FreeBSD system and kernel building

FreeBSD 6.1 is installed with its kernel sources. KDE desktop environment is chosen, Firefox as Internet browser, VIM as editor and KDevelop as development environment are installed, too. The following tools are installed separately:

- autoconf
- automake
- libtool
- cvs
- svn

The latest Ethereal version (v.0.99) is obtained from the SVN (subversion) server. For configuring the X server, the instructions of the FreeBSD handbook (http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/x-config.html) are used.

To use a current SCTP kernel implementation with the latest and bug-fixes, the kernel needs to be patched and built again.

The patches are obtained from Randall Stewart, who kindly created an account for his the CVS server:

```
# cd /root/sctp_kern
# cvs -d :ext:<user>@stewart.chicago.il.us:/usr/sctpCVS co KERN
# cvs -d :ext:<user>@stewart.chicago.il.us:/usr/sctpCVS co APPS
```

The included setup program was executed:

```
# cd KERN
# ./setup_FreeBSD_src.sh
```

Further on the kernel configuration file needs to be modified by these lines:

```
options SCTP
options SCTP_DEBUG
```

The kernel has to be built and installed:

```
# cd /usr/src
# make buildkernel KERNCONF=<config-file>
# make installkernel KERNCONF=<config-file>
```

General instructions about configuring and building of FreeBSD kernels can be found at (http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/kernelconfig.html).

Finally four files need to be copied to other locations:

```
# cp /root/sctp_kern/KERN/netinet/sctp.h /usr/include/sys
# cp /root/sctp_kern/KERN/netinet/sctp_uio.h /usr/include/sys
# cp /root/sctp_kern/KERN/freebsd6_1/sys/socket.h /usr/include/sys
# cp /root/sctp_kern/KERN/freebsd49/netinet/in.h /usr/include/sys
```

To be able to use the corresponding library, it has to be built:

```
cd /root/sctp_kern/KERN/usr.lib
gmake
cp libsctp.a /usr/lib
ranlib
```

Code that uses the SCTP library needs to be compiled like this:

```
gcc terminal.c -lsctp
```

Statutory declaration / Eidesstattliche Erklärung

I have prepared this thesis entirely by myself using only the sources mentioned. This thesis – or any variation thereof - has never been submitted to any examination authority.

Die vorliegende Arbeit wurde von mir selbstständig und nur unter der Verwendung der angegebenen Quellen angefertigt. Die Arbeit hat in dieser oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

_____

(Michael Nordhoff)

Essen, 7th of September 2006 / Essen, 7. September 2006