Design and implementation of a test scenario to evaluate end-to-end security solutions for SCTP

A project seminar report

Michael Nordhoff

Institute for Experimental Mathematics
Computer Networking Technology Group
University of Duisburg-Essen, Germany

May 2006

Abstract

The Stream Control Transmission Protocol (SCTP) is a new reliable transport protocol. Although it was 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 in modern network topologies is an advantage of SCTP compared to the traditional protocols. But one key for the success of SCTP will be the ability to secure the transported data. Services like authentication, authorisation and confidentiality are essential and must be provided for SCTP traffic. There are already three important approaches, which all have their own advantages and drawbacks. This seminar work explains briefly SCTP and the security solutions, summarises the functional and performance-related drawbacks and introduces a testbed to measure the performance of these security solutions. In a following study work tests will be performed to evaluate the performance-related drawbacks which are up till now theoretically deducted but not comprehensively confirmed and benchmarked with real test scenarios.

Contents

1 Introduction

1.1 Motivation

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 transport protocol Stream Control Transmission Protocol (SCTP) with its applications and implementations.
A problem of the acceptance of SCTP in a common way 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 that decides if a solution will be accepted or not. 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 denial-of-service attacks it is more invulnerable and therefore more reliable and secure.
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 security 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 the testbed presented in this work.

1.2 Overview of SCTP

1.2.1 Introduction into SCTP

The Stream Control Transmission Protocol is a 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 network protocol to send packets to and receive packets from the peer instance. 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 SIGTRANS working group of the IETF to gain a possibility 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 especially in this area of networking. In the end the outcome was a standard of a multi-purpose transport protocol which was defined in the RFC 2960 and is renewed by additional functions described in some RFCs and Internet-Drafts ([6], [9], [10], [11]).

1.2.2 Packet format

Before we give a detailed introduction to SCTP some SCTP-related technical terms have to be introduced, because their meanings are not exactly equal to 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.

The rough format of a SCTP packet is quite simple: it contains an SCTP common header and one or more chunks grouped behind the header – as shown in figure 1. The common header consists of the source port number and the destination port number (each 2 bytes). 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 came out that this is a week algorithm and was changed to a CRC32 algorithm [6].

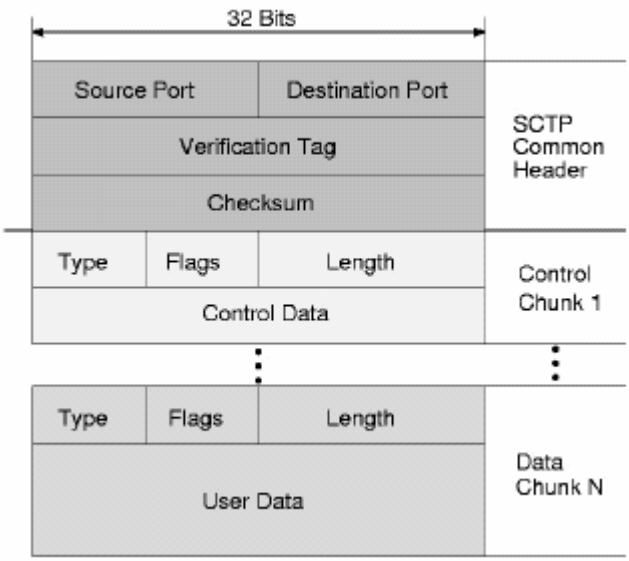

Figure 1: Basic format of SCTP packet [12]

Chunks are designed in a way that they are fully self-descriptive and have 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; 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 at the beginning of the packet before DATA chunks.

The chunk length can vary, but must complete a 32-bit word. The last word must probably be filled up with padding bits. Each chunk has got a header which uses the first 32-bit word. The first byte sets the chunk type. The value 0 defines a DATA chunk. There were initially 14 other chunk types defined which are all control chunk types (see table 1). The next byte contains the chunk flags. The meanings of the flags are related to the different chunk types. There are no common flag definitions. Byte number 3 and 4 contain the length of the complete chunk except the padding bytes at the end.

ID Value	Chunk Type
0	Payload Data (DATA)
1	Initiation (INIT)
2	Initiation Acknowledgement (INIT ACK)
3	Selective Acknowledgement (SACK)
4	Heartbeat Request (HEARTBEAT)
5	Heartbeat Acknowledgement (HEARTBEAT ACK)
6	Abort (ABORT)
7	Shutdown (SHUTDOWN)
8	Shutdown Acknowledgement (SHUTDOWN ACK)
9	Operation Error (ERROR)
10	State Cookie (COOKIE ECHO)
11	Cookie Acknowledgement (COOKIE ACK)
12	Reserved for Explicit Congestion Notification Echo (ECNE)
13	Reserved for Congestion Window Reduced (CWR)
14	Shutdown Complete (SHUTDOWN COMPLETE)
15 to 62	reserved by IETF
63	IETF-defined Chunk Extensions
64 to 126	reserved by IETF
127	IETF-defined Chunk Extensions
128 to 190	reserved by IETF
191	IETF-defined Chunk Extensions
192 to 254	reserved by IETF
255	IETF-defined Chunk Extensions

Table 1: Chunk Types

Each chunk can have permanent chunk parameters and optional chunk parameters (3.1.5 of [3]). Permanent chunk parameters are defined for each chunk type and do not need a generic format. Optional chunk parameters are placed behind the permanent ones – in case they are used. This optional parameters use a minimum of two 32-bit words. The first 2 bytes set the parameter type, the following 2 bytes the parameter length. After that the parameter data follows.

A usual data chunk contains the following fields:

The first word builds the chunk header as described above. The chunk type value is 0x00. Three flag meanings are defined:

- The U bit is set to 1, if the chunk contains user data which is unordered.
- The B flag is set, if this chunk contains the beginning of a user message.
- The E flag is set to 1, if the chunk contains the ending part of a user message.

Thus one or both of the B and E flags is 0, if the user message is fragmented. In this case the peer holds this data and waits until it can reassemble the message. The next word after the header is the transmission sequence number (TSN). Each DATA chunk of an association gets

an incremental TSN by the sender. It is used for reassembling fragmented messages and for reliability. It can be checked if a chunk is missing.
The next word contains the Stream Identifier (SI) and the Stream Sequence Number (SSN). The Stream Identifier allocates the DATA chunk to a certain stream, the Stream Sequence Number is an incremental number which counts the messages sent in this stream.
The next 4-byte field is the Payload Protocol Identifier. This field is set by the ULP and is just transmitted unchanged to the peer by SCTP. The ULP may use it to find out what kind of data is coming in. It can also be used for packet filters or network monitoring.
After that the ULP payload is attached.

1.2.3 Main features

SCTP has got a lot of essential features. Up to now some of them are unusual in a general-purpose data transmission protocol. Others are analogue to well known features of traditional protocols. The collection of the different features as a whole characterises SCTP. Its main features are presented in this chapter:

- **Multi-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 is known from TCP.
- **Multi-homing:** SCTP endpoints are able to have several network interfaces with one IP address each. During initiation of an association the 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 [12]). This feature makes it possible to obtain a high reliability and robustness against single interface and network failures. The 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.
- **Flexible delivery:** If DATA chunks of one stream arrive at a peer in the wrong order – i.e. the next message of the sequence is missing so far – the other chunks need to be stored in the resequencing buffer (see p. 38 in [12]). In case a message does not need to be in the order of a sequence, the U flag (unordered flag, see above) 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.

- **Bundling:** 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 both 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 put it in a single DATA chunk and into an 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 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 transmission protocol that 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.
- **Flow and congestion control:** The flow and congestion control of SCTP is very similar to the TCP ones ([13], [14]). 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 slow; 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 MTU and the transmission continues at a lowest level with the slow start phase.
- **SCTP association management:** SCTP is a connection-oriented protocol; the association must be set up, maintained and shut down. For these purposes procedures are exactly defined. Figure 2 shows the complete state machine. It is described in RFC2960.

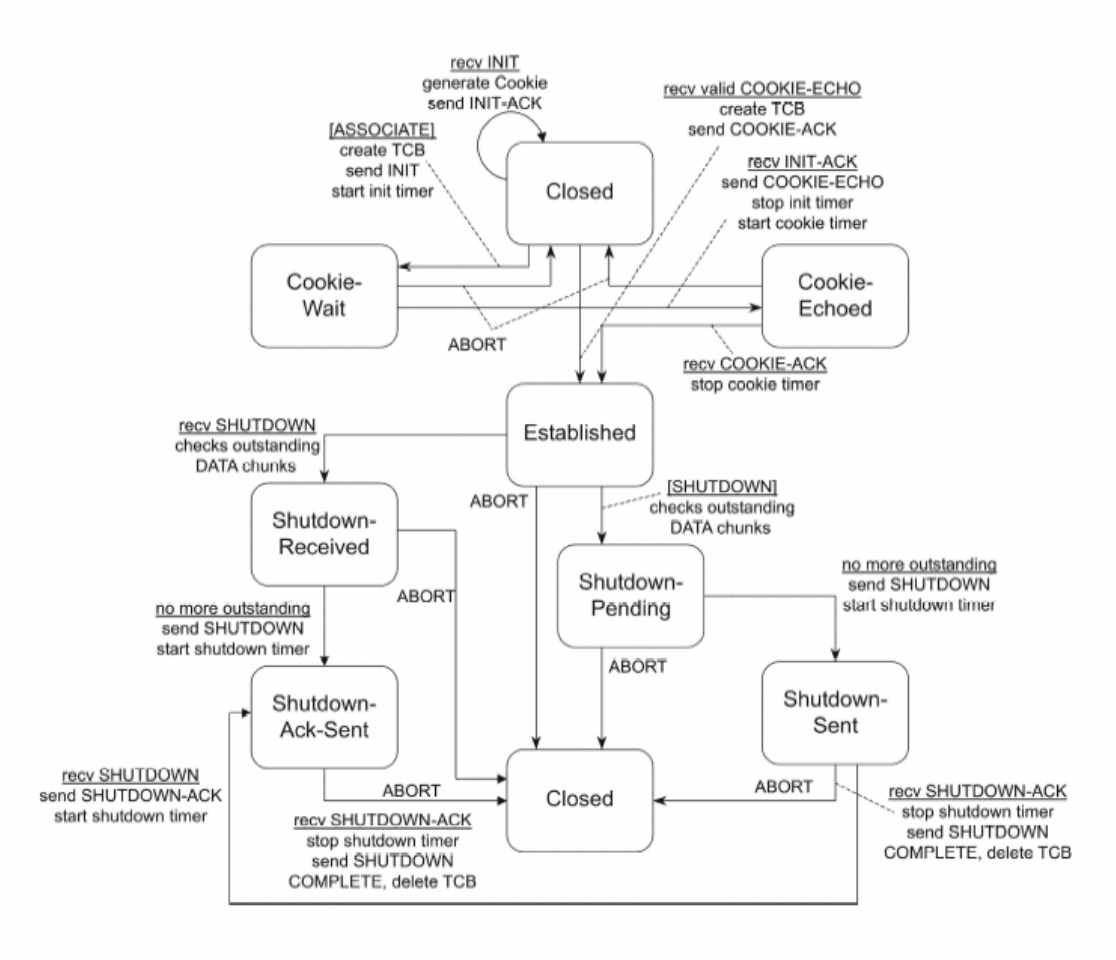

Figure 2: SCTP's association state diagram

Association setup:

Similar to TCP also SCTP uses a handshake procedure for setting up the association, but unlike TCP it performs a four-way handshake instead of a three-way handshake (see figure 3). First endpoint A sends an INIT chunk, it includes an initiation tag which is used later for the common header. This simple feature prevents the peers from blind attacks. A SCTP packet which has a verification tag not belonging to an established association is discarded immediately. Some other parameters, like the initial Transmission Sequence Number (TSN), the Advertised ReceiverWindow Credit (a_rwnd) or the list of the endpoints IP addresses are also contained in the INIT chunk. Endpoint B receives the INIT chunk and responses with an INIT-ACK chunk, which contains the same fields as the INIT chunk. Additionally it contains a cookie with all information of these fields sent by the two endpoints, a time stamp and a secure hash value generated by a secret key of endpoint B. Because of this cookie endpoint B can discard all data about this initiation and does not need to allocate resources for the potential association. An attacker cannot use init-flooding for denial-of-service attacks against SCTP – like syn-flooding in the case of TCP. Endpoint A sends the state-cookie back in the COOKIE-ECHO chunk. Endpoint B checks the originality and the integrity of the cookie and sends a COOKIE-ACK chunk. Usually both endpoints have now reached the established state and can start with transferring user data.

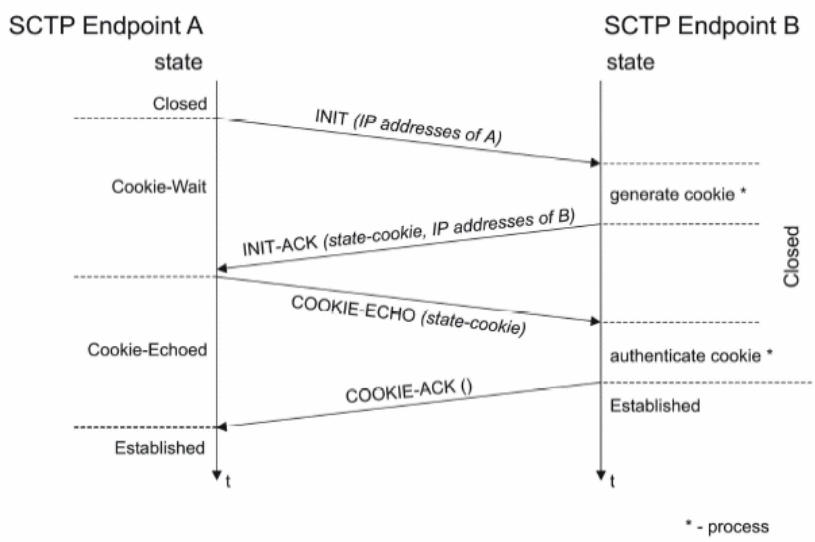

Figure 3: Association setup

Association shutdown:
There are two ways to shutdown the association: the so-called “graceful shutdown” and “abortive shutdown”. The graceful shutdown is usually initiated by the ULP. SCTP performs a three-way handshake, SHUTDOWN chunk, SHUTDOWN-ACK chunk and SHUTDOWN-COMPLETE chunk are send like it is shown in figure 4. During ‘shutdown pending’ state and ‘shutdown received’ state of the respective endpoints remaining data chunks are sent and the peers are waiting for outstanding acknowledgements.

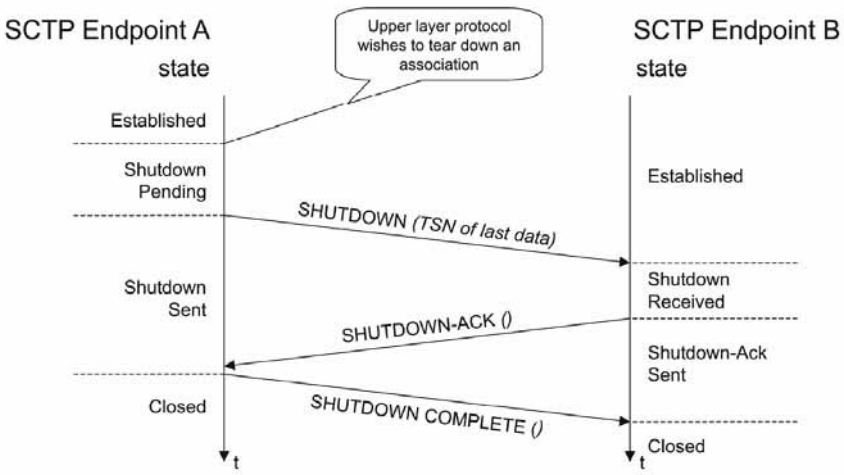

Figure 4: Graceful shutdown

The abortive shutdown is an unreliable best-afford shutdown to let the peer know that the association is stopping. It is used by the application or directly by SCTP. The endpoint who wants to stop the association sends an ABORT chunk to the peer and enters the ‘closed’ state. The peer endpoint which receives the ABORT chunk verifies it and also enters the ‘closed’ state.

- **Path and peer monitoring:** SCTP endpoints monitor all paths to the peer endpoint of an association. HEARTBEAT chunks are sent regularly over all paths except the primary path. Each HEARTBEAT chunk needs to be acknowledged by a HEARTBEAT-ACK chunk. A path where no heartbeat acknowledgement takes place in a certain time gets the state 'inactive'. The number of events where heartbeats were not acknowledged within a certain time, or retransmission events occurred is counted on a per association basis, and if a certain limit is exceeded, the peer endpoint is considered unreachable, and the association will be terminated [15].
- **SCTP extensions:** After the initial definition of SCTP in RFC 2960, SCTP has been continuously developed and some extensions have been proposed in internet drafts. These extensions may be important for the acceptance of SCTP in future scenarios.
  - **Partial reliable delivery** is a feature that allows ignoring loss, unordered or late DATA chunks in case it is wanted. This is useful for real time applications where audio or video data is not usable if it is late. This feature is realized with the help of a new (optional) parameter type, which can be used in the INIT and INIT-ACK chunks to make sure both endpoints support this feature. In this case a new control chunk called FORWARD-TSN can be sent to perform the partially reliable service. Without this extension the sender has to retransmit a message until it is acknowledged – no matter if it is useful anymore or not. Retransmissions are performed, bandwidth is used and (more) congestion can be caused even when a package will not be used by the receiver's ULP anymore. With the partial reliable delivery extension timed reliability is possible even in case a message already has got a TSN or has been transmitted or retransmitted without success. Before transmitting or retransmitting a message that has already got a TSN, the lifetime is checked by the SCTP sender. If the lifetime has expired, the sender is in charge of sending a FORWARD-TSN to the receiver. It contains a new cumulative TSN number. The receiver considers any missing TSNs earlier than or equal to this value as received, and does not report them as gaps in subsequent SACKs anymore. Recently it got the status of an RFC [9].
  - **Dynamic address reconfiguration** is an extension that allows the reconfiguration of IP addresses of an already established association. Two new control chunks (address configuration change ASCONF and address configuration acknowledge ASCONF-ACK) allow adding and dropping of IP addresses. Both associated endpoints can send a request to add or to drop an IP address. Also the primary path can be changed. This feature is very useful in mobile networks where endpoints move between ranges of different networks. The mobile device can use the IP address of the current network and drop the IP address of the former network which is currently unreachable. The association keeps established all the time [10].

1.3 Security Solutions for SCTP

1.3.1 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 unidirectional, 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 5 shows a block diagram with the most important IPsec-related RFCs .

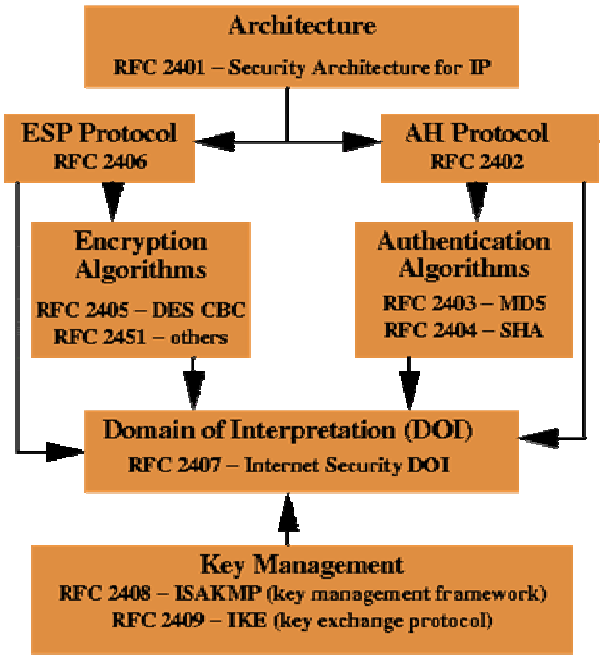

Figure 5: IPsec protocol and document overview [16]

It is possible to combine IPsec with SCTP. At first view 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 mulithoming 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 ([8]) 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 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.

1.3.2 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 (SecureSHell, SecureFileTransmissionProtocol, …) were developed. Because 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.
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.

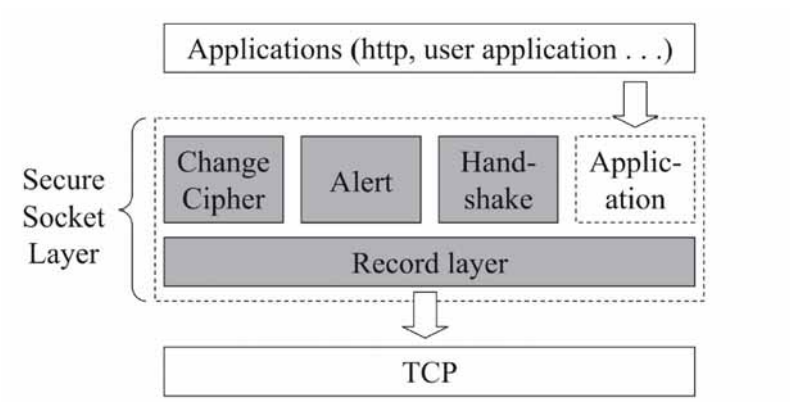

Figure 6: SSL protocol overview [12]

For using TLS with SCTP there is already a standard specified (RFC3436 [7]). 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 7 shows a scenario where 3 applications use the same SCTP association. Each application uses one stream. Application number 2 uses plain SCTP. Application 1 and 2 use secured transmission by setting up a TLS session in their streams. The advantage is the possibility to mix 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 harm 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.

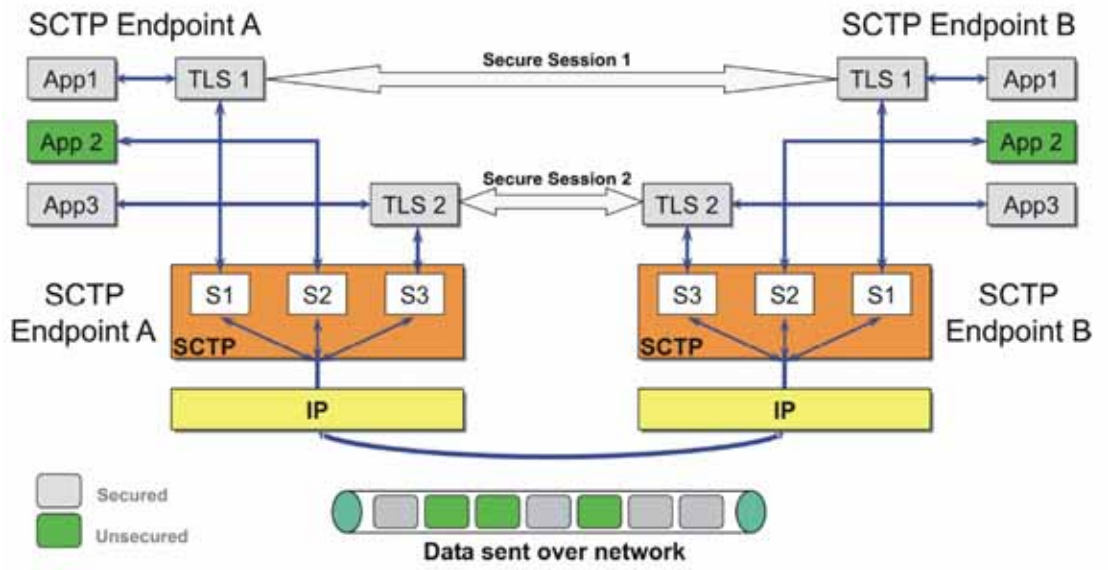

Figure 7: Secured and unsecured transmission using TLS over SCTP [12]

1.3.3 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 ([12], [17]). It is downward compatible to plain 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 is 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 8.

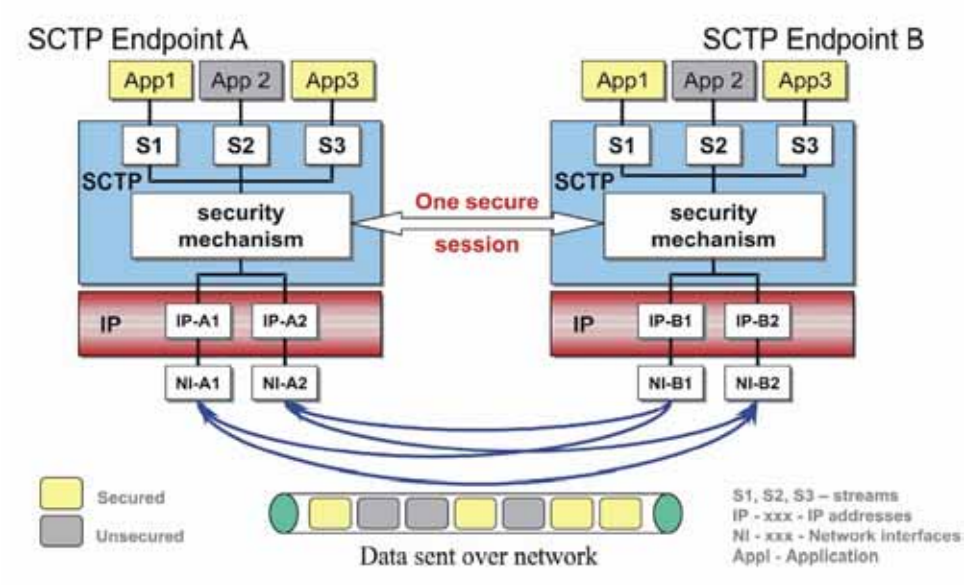

Figure 8: Secured and unsecured transmission using S-SCTP [12]

S-SCTP offers a set of security levels, which can be changed during a secure session lifetime. 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 good as or even better than the other two security solutions.

2 The test scenario

2.1 Purpose

In chapter 1.3 we presented the three standard security solutions for SCTP. Because IPsec and TLS were not developed for SCTP, these security solutions have limitations and problems. The table 2 provides an overview of all three security solutions. They are 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 multihoming	+	(-)	+
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 [18]

There are *functional* limitations due to new features introduced by SCTP, which cannot be supported by some standard security solutions. These limitations can be found in the table 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. For the same reason the proposed 'Partial reliable delivery' extension is not supported by TLS over SCTP.
A functional security problem of TLS over SCTP is the lack of protection of the SCTP control chunks. Because TLS just protects the user data which happens on 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 the table above:
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.

The multi-homing feature is not well supported by the **SCTP over IPsec** solution. Security Association (SA) management and key management of IPsec were developed to cope with unique IP addresses for each peer. Although there is a new standard allowing SAs with more than one IP address per peer, there are no adequate implementations available until now [12] [8]. The management of the security sessions in general is problematic with IPsec. Even in case implementations would be available, time and effort might be significant. Nearly the same problems occur with the 'Dynamic address reconfiguration' extension in combination with IPsec. As an extension of the multihoming feature it would increase the complexity to manage the security associations. 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 packages need 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 just identified theoretically. 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 test scenario and the design of the testbed is to measure these restrictions of the different security solutions which are caused by performance problems.

2.2 Design

The performance problems of the different solutions are presented in chapter 2.1. The testbed must be able to deliver results for all features mentioned above. The only feature that affects the hardware setup of the testbed is multihoming which requires that each endpoint has at least two NICs. We need two computers working as two endpoints in different IP networks. A third Computer will serve as a router. The router has got four NICs, each connected directly with one peer's interface (see figure 9).

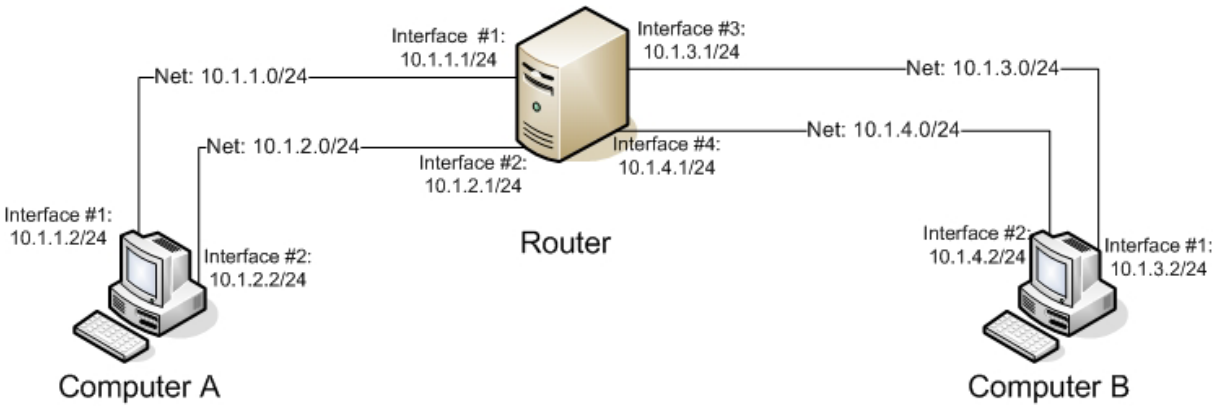

Figure 9: Testbed design

These two endpoints satisfy the multi-homing requirement. All other features only affect the software configuration. With the testbed we want to see the differences and similarities of the three security solutions and as a reference we use plain SCTP without security features. All solutions will provide multiple streams; this SCTP core feature is actually supported by all

solutions. The overhead of small messages (when bundling happens) and the overhead for long messages (when fragmentation takes place) can be measured when different user data frame sizes are generated at the endpoints.
The management of the security sessions is in case of S-SCTP or TLS over SCTP the task of the own protocol implementations. In case of SCTP over IPsec we need the IPsec implementation at the endpoints to perform transport mode. For key and session management we will use the tools provided by the IPsec implementation.
The Partial Reliable Transport is an SCTP extension, which has recently been implemented into the sctplib version which we used. Dynamic Address Reconfiguration is only implemented in a test version of the sctplib and could be used by activating and deactivating one network interface by the operating system during an association lifetime.

The tests should be performed in several environments which are in a way similar to those in real applications. This is the reason why we need different links between the communicating nodes. Essential link properties are mainly the bandwidth, the delay, loss, jitter and unordered delivery. All these can be simulated by software which needs to run on the router between the endpoints. Before forwarding a package the software decides if the package should buffered for a certain time or even should be dropped. All these features are provided by the software *'dummynet'* in connection with *IP firewall (ipfw)* which needs to be installed at the router's system. More details can be found in the next section.

2.3 Implementation

2.3.1 Hardware

The two peer computers are standard desktop computers with AMD Athlon XP 2000+ processors, 512 MB RAM each, and an onboard 100Base-T network interface card which is used to connect to the internet for remote control, installation and management. Both computers have additionally 2 PCI gigabit network interface cards.
The router is also a personal computer. It is a 64bit system from AMD and has got 1 GByte RAM. The onboard 1000Base-TX network adapter is connected to the internet. Additionaly two 64bit PCI gigabit network interface cards with two interfaces are used. These network interfaces are directly connected with usual category 5 network cables. To avoid a possible bottle neck at the routers site we connect both interfaces of each PCI64 card of the router with the two interfaces of the same peer (see figure 10). This configuration makes sure that just one SCTP primary path with high traffic load will go through one network card.

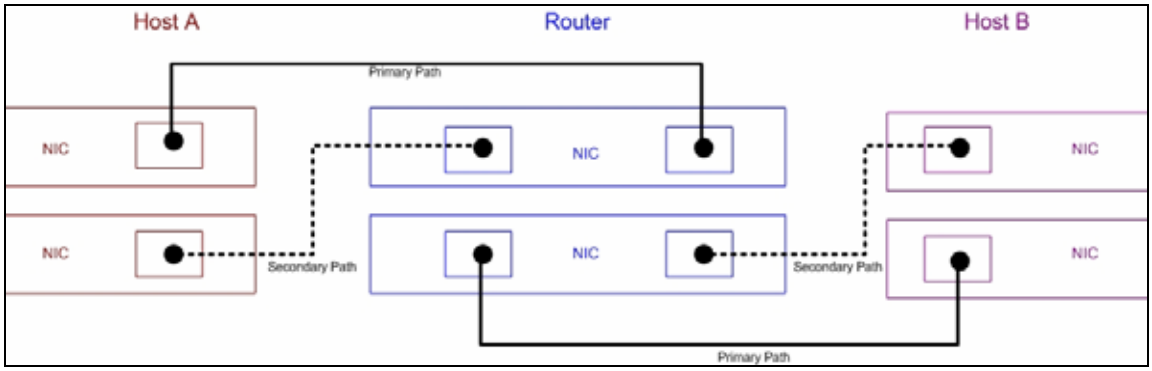

Figure 10: Network interface allocation to avoid bottlenecks

2.3.2 Operating Systems

The peer computers are using the following operating system:
SuSE Linux distribution version 9.1 with kernel 2.6 and the KDE window manager system version 3.2.
During the installation the following selections were chosen:

- 'KDE complete'
- 'C/C++ Compiler and Tools'
- 'Network/Server'
- 'KDE Development' and
- 'Experienced User'.

Definitely not all packages of these selections are needed, but at least a few of each are recommended to install.
The operating system of the router is FreeBSD. This decision is made because of *dummynet*, which is part of FreeBSD and can be used for simulating different links. Precisely FreeBSD 5.4-REALEASE AMD64 is installed. Because we only need some basic routing functions and the *dummynet*, we only had to install the following two packages:

- 'Base'
- 'Developer'

2.3.3 Network Setup

We have four IP subnets for our testbed (without the configuration to the internet) as shown in figure 9.
The router can be configured with the configuration tool 'sysinstall' or directly by modifying the /etc/rc.config-file. To enable the routing function, the option 'gateway_enable' must be set to 'yes' (see Appendix B for rc.conf example).
For setting up the IP configuration of the peer computers the easiest way is to use YAST which writes one configuration files for each interface. If you make manual changes, make sure that it will not be overwritten later on by YAST scripting.

2.3.4 SCTP Software Setup

We use the userland implementation developed by a cooperation of Siemens, the Computer networking technology group of the University of Duisburg-Essen and the Münster University of Applied Sciences. The SCTP libraries and the corresponding socket API can be downloaded from the following web site [19].
The newest version is located on an SVN server of the Institute for Experimental Mathematics (IEM), which we used for our testbed. After downloading and unpacking or checking out (see instructions in Appendix B) it needs to be compiled and installed. The glib library location has to be specified as a parameter when the configure script is executed. The glib version 1.2 is required. Compiling and installing works with the help of make. Important is that you need root rights for the final installation (see Appendix B).
For checking the success of the installation and the functionality of SCTP with the network there are some simple but helpful programs that come with the sctplib. For example an echo-server can be used with a terminal program (example in Appendix B).

2.3.5 SCTP over IPsec setup

Setting up the testbed for the SCTP over IPsec security solution we do not have to modify anything but the IP connection. SCTP remains unchanged.
For IPsec we choose the transport mode because just two endpoints are involved. For getting confidentiality of the payload we need to use the Encapsulated Security Payload (ESP) protocol, which encrypts the data, unlike the Authentication Header (AH) protocol, which ensures only authentication and integrity.
To set up IPsec connections we make sure that the ipsec-tools package of SuSE is installed. 'setkey' is a tool to manage the security association and security policy databases. 'racoon' is a daemon to manage IKE keys.
For our purpose it is adequate using pre-shared keys. Each peer needs 3 files: the key file (e.g. /etc/psk.txt) where the pre-shared keys of each remote IP address are stored. Then we need a configuration file to set the security policies of all possible associations with the help of *setkey*. Furthermore there is a configuration file for starting the racoon daemon to provide the keys. To start the IPsec associations setkey must be executed and racoon must be started – both with the corresponding files. See [20] for general configuration help and Appendix B for the specific files and commands.

2.3.6 TLS over SCTP setup

To run TLS over SCTP the openssl package must be changed. In the beginning we need the code of openssl version 0.9.7c. This can be downloaded from [21]. After running the configure script a few code files have to be exchanged to force openssl to use the SCTP socketapi instead of the usual TCP/IP interface. We can reuse the code files already manipulated by Unurkhaan [12]. The code files 'bio.h', 'bss_sock.c', 'ssl.h' and 'ssl_lib.c' need to be substituted. The perl package version 5 needs to be installed. This can easily be done with YAST. Also the Makefile need to be substituted and changed: system-specific paths of some libraries have to be adapted; note that partly the libraries of S-SCTP are used. Additionally the path of perl is version-related and must be adapted.
Furthermore the changed bss_sock.c and ssl.h files must be adapted (see Appendix B). After changing all these files 'make' and 'make install' can be run.

2.3.7 Secure-SCTP setup

The S-SCTP userland software is a modification of the sctplib userland prototype implementation described in [12]. It contains three new modules integrated into the common SCTP protocol architecture: the crypto controller is an additional module inserted in the control path, the encryption/decryption module and the authentication module are located in the assembly/disassembly module of the data path (see figure 11).

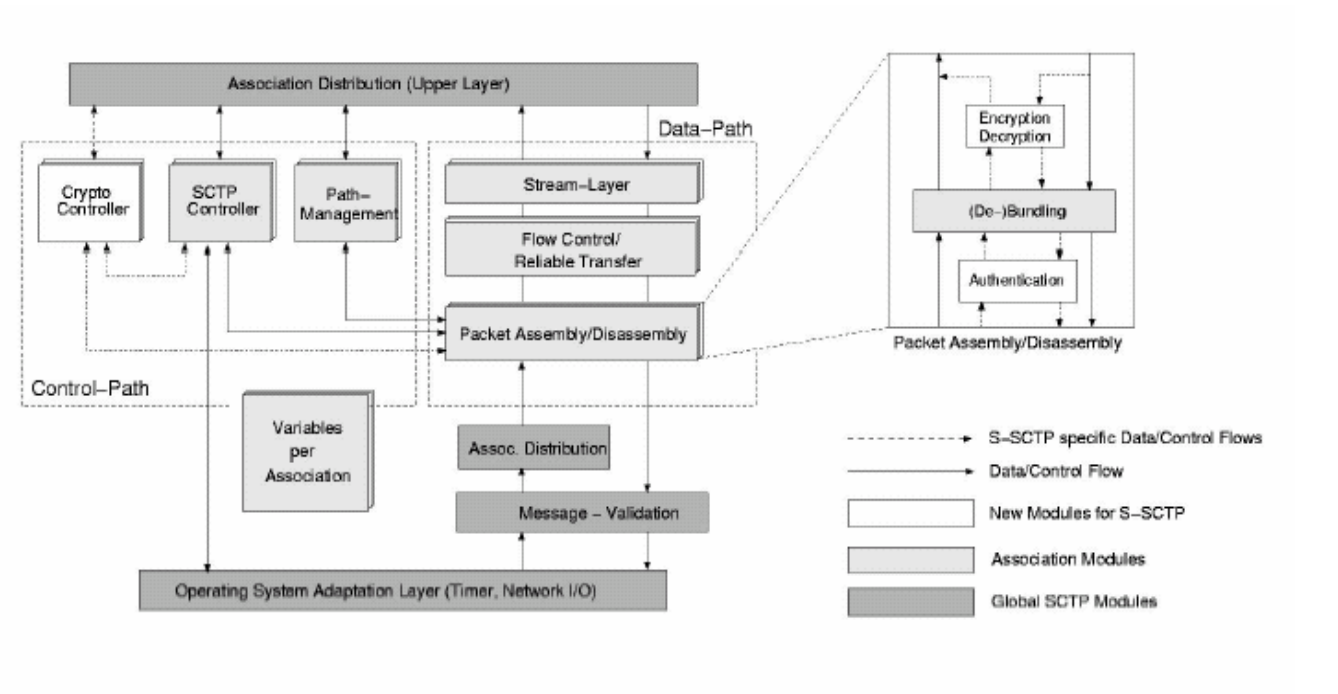

Figure 11: S-SCTP protocol architecture [17]

The additional functions of initiation, deletion and rekeying are declared in the file *ssctp.h*, en-/decryption and authentication functions are declared in the file *cryptocrtl.h*. The header file for the new data structure used in secure sessions is *ssctp_messages.h*. For setting up the implementation of S-SCTP we need to configure, compile and install the extended SCTP libraries, which were packet in the file *ssctp-stabile.tar.gz*. After running the configure script (again with the glib prefix), the *Makefile* file needs to be adapted before compiling as usual (see Appendix B).
Also the socket API was adapted. The user can call the functions of it like the functions of the standard SCTP socket but can also set an encryption flag. The S-SCTP socket API uses the new functions of the extended sctplib mentioned above.
The file *socketapi-sec.tar.gz* needs to be unpacked and configured as usual. I copied a few header files to the paths which will be accessed during compilation (see Appendix B).
After this is done the new socket API can be compiled and installed.

To perform S-SCTP transmission I also got a server-client-application written in C++ by Unurkhaan. To install the software you need to unpack the file *test.zip* and adapt the *Makefile* file (Appendix B). After compiling with the help of make, it can be used to transfer data bidirectionally. The number of streams, the framesize, port numbers, source and peer addresses, and number of loops are some of the important parameters which can be specified in the command line (see Appendix B).

2.3.8 Ethereal Setup

For analyzing the bits on the wire Ethereal should be installed. This program is an open-source, graphical and approved protocol analyzer which is also included in the SuSE distribution. SCTP is already integrated, but if we want to analyze the SCTP payload of TLS over SCTP or S-SCTP-specific chunks, an modified version of Ethereal must be installed.

The program files were extended by Unurkhaan (see chapter 6.2 of [12]). Installing this version is straightforward (Appendix B).

2.3.9 Ipfw/Dummynet setup

The kernel modules 'ipfw' and 'dummynet' are used to simulate different connection links. Ipfw can be permanently loaded by an entry in the /etc/rc.conf file, dummynet can be permanently loaded by an entry in /boot/loader.conf (Appendix B). Also the security level has to be decremented (see also rc.config in Appendix B) to a negative value to allow firewall manipulation while the system is running.
Setting bandwidth limitations, loss, delay or even jitter with IP firewall and dummynet need to be done with the firewall control program 'ipfw'. Rules for the firewall need to be added to the rules table. For our purpose the rules mainly put the packages into so-called pipes. These pipes get configured also with the help of 'ipfw', for example bandwith limitations can be allocated to a pipe (if dummynet is loaded). A script for configuring dummynet is attached (Appendix B).

2.3.10 Perftool

Another tool to measure performance characteristics of SCTP is called *Perftool*. It is written in C++ and developed in the Computer Networking Group of the IEM. It uses *sctplib* and *socketapi*. For configuring and compiling the usual commands have to be executed like it is mentioned at the other program's installation guidelines. The program can generate output files for statistical usage with the help of *R*, a free software environment for statistical computing and graphics [22]. It is also offered as RPM to install with SuSE's *yast*. For displaying the statistical results, *gnuplot* need to be installed; it comes with the SuSE distribution and can also be installed with *yast*.
Perftool needs to be installed at the sending host and at all peer hosts. The sender can be executed in a way that it sends data with a certain frame rate or as fast as possible to determine the throughput. A list of peer hosts can be passed for one test scenario. Different frame sizes, run times, number of runs, and a start-up time without measuring can be set with parameters. The exact command can be found in Appendix B. The measured and released results are written as a vector in a file. Calculations result in scalars which describe statistical data like the maximum, the minimum or the average of delay, jitter, congestion window size, receive window size or round trip times.
With the help of scripts, which are also in the *perftool* folder, graphs can be displayed or written in portable document format (PDF) files.

3 Results

After setting up the testbed, installing and configuring all software, all security solutions are ready to use with the programs mentioned. The data transfer with its throughput rates and other values is displayed by the programs. To ensure that the transfer works properly like it is expected we monitor the bits on the wire with *ethereal*. We need to make sure that all security features are actually performed like intended. Also the features like multihoming, multi-streaming, bundling and fragmentation can be observed by *ethereal*. The functionality of *dummynet* also needs to be confirmed if it works like intended.

3.1 Confirmation of the security solutions

With *ethereal* data transfer of all security solutions was checked, because ethereal allows exact analysis of the transmitted bits. All important information concerning the network and the transport layer is displayed.
As an example the following figure (figure 12) shows the handshake and user data transfer of a data transfer using TLS over SCTP. User data is encrypted as intended. After the usual SCTP handshake is done, the TLS handshake is reported. The following application data is embedded in the TLS record layer protocol – in an encrypted way.

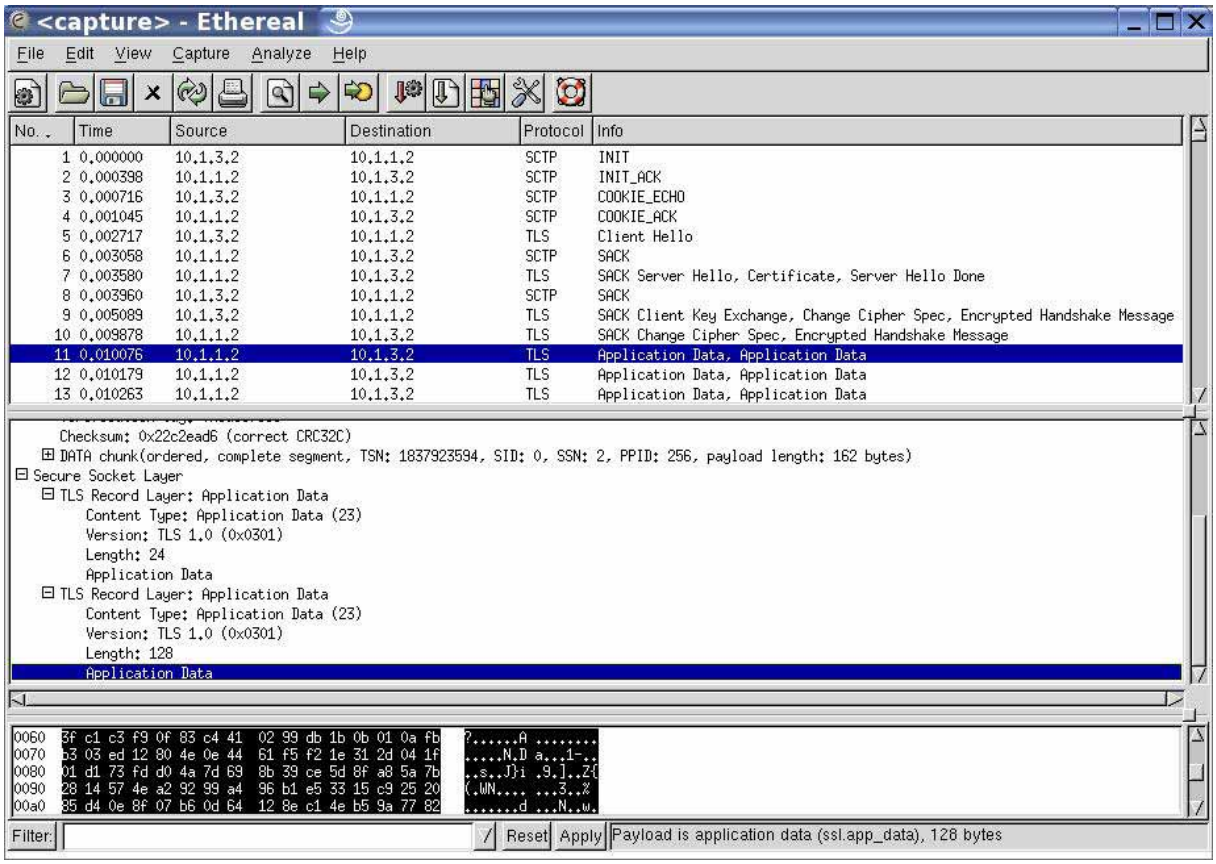

Figure 12: Ethereal snapshot – TLS over SCTP

Data transfer using S-SCTP was verified in the same way. The new chunk type EncData is displayed and also the fixed-sized AUTH chunk at the end of the SCTP packet can be seen (figure 13).

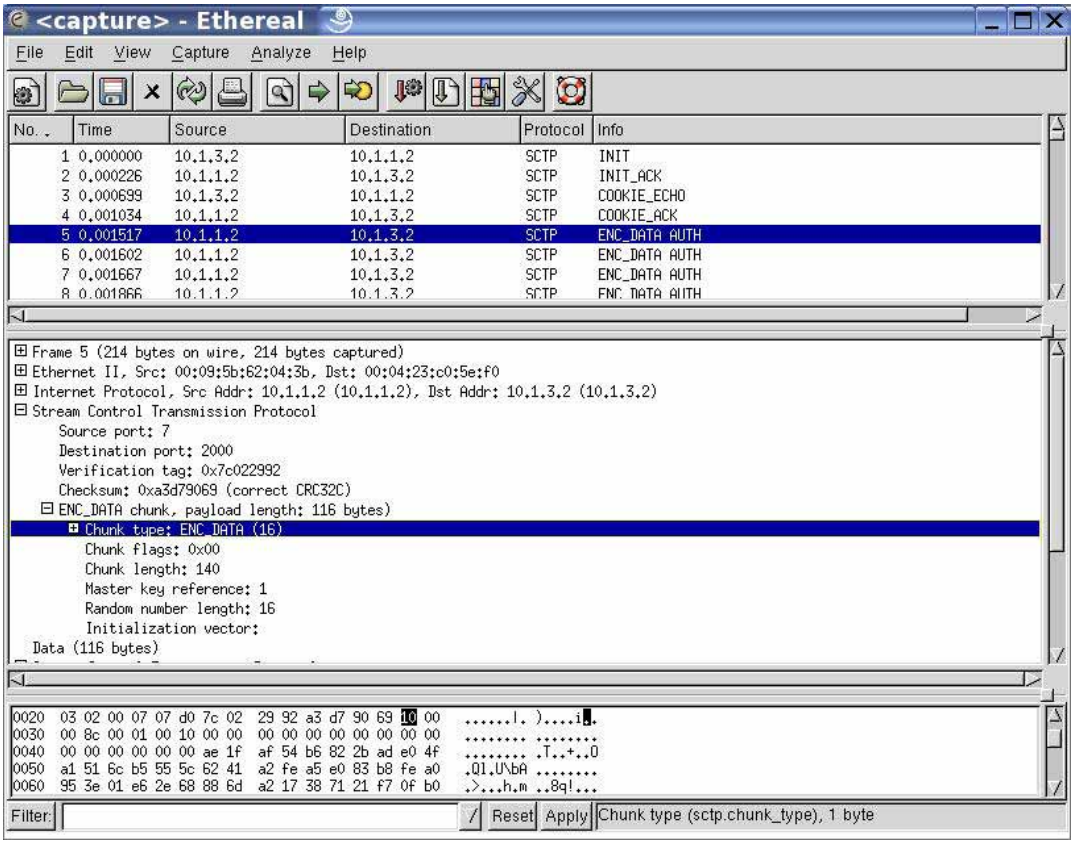

Figure 13: Ethereal snapshot – Secure-SCTP

Another example is shown in figure 14: the initiation and data transfer using SCTP over IPsec. The hosts negotiate (with the ISAKMP protocol) the security standards which will be used for the association. The pre-shared key algorithm is chosen, ‘quick mode’ is displayed. The following data packages are encrypted. Only the header of the network protocol is in plain text; at the transport layer the data is already encrypted, that is the reason why no SCTP-specific information can be seen.

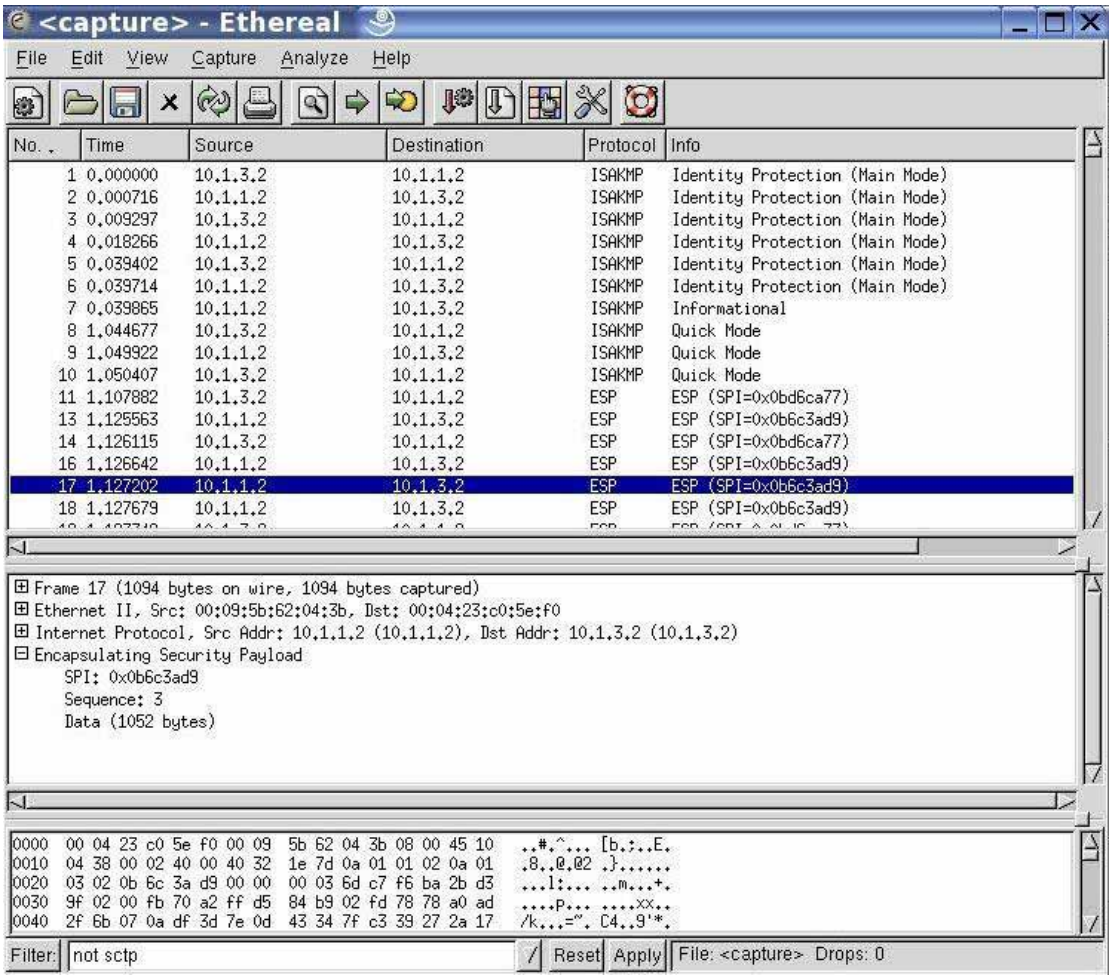

Figure 14: Ethereal snapshot –SCTP over IPsec

3.2 Usage of SCTP features

Using **multiple streams** for transmission is provided by the *sctplib* and the test programs. This was checked by observing different stream numbers of the data chunks which were captured by *ethereal.*

The **multihoming** feature works as specified: during the handshake all possible IP addresses were exchanged by the endpoints. When an association was established, HEARBEAT chunks were observed at all backup paths with the help of *ethereal*. When simulating a lossy link the retransmits were sent over the non-primary path like it is defined by the specifications.

Bundling was also observed when transferring small frames. Many data frames were put into one SCTP packet. **Fragmentation** could also be observed when exceeding the MTU by long data frames.

3.3 Checking the link simulation

With the help of the data sending and receiving programs and a tool at the router the overall data throughput of the links can be measured. Delay, jitter and loss can be observed with tools like *ping* and also with the *ethereal* program.
The overall data throughput can be measured simply with the programs *ipband* or *iptraf* or any other traffic monitor program. The programs show the current throughput passing the interfaces. The throughput remained at the bandwidth that was set by *dummynet* or remained slightly under it.
Delay can be simply measured with *ping*. With the help of the ICMP packets ECHO_REQUEST and ECHO_RESPONSE the program calculates and prints the round trip time (RTT). When setting a delay with *dummynet*, the RTT increases by two times the specified delay apart from inaccuracies of much less than 1 millisecond. The probability and random functionality can also be checked by *ping*. After sending and receiving a high number of ICMP packets, the ping summary shows the percentage of received and lost packets when terminating the program. Jitter effects can be obtained by using the random function with different pipes which have different delays. Also this can be monitored and validated with the output of *ping*. If the jitter is high enough, unordered delivery is the result.
All these samples can also be validated with *ethereal*. It protocols all packets on wire with the exact time and size, hence complex statistics can be calculated.

4 Conclusion and future work

This work deals with the setup of a testbed to compare different security solutions for SCTP. First, chapter 1.2 gives a short introduction into SCTP and highlights the differences to TCP. In the next chapter, three security solutions for SCTP are presented and evaluated. The standard security solutions TLS and IPsec are developed for TCP and hence have functional limitations and can not support all SCTP features. This is the reason why S-SCTP was developed and implemented. S-SCTP can be assumed to be an optimal solution because it integrates the security functionalities directly into SCTP. Chapter 1.3 begins with a comparison of these security solutions and shows the strengths and weaknesses of each solution. The functional limitations of TLS and IPsec can not be changed, without the modification of the protocol specifications. But these solutions do not only suffer from functional limitations, but also from performance related ones. The purpose of the testbed developed in this work is to measure these performance limitations, in order to be able to quantify them. So in a first step during the development of the testbed, a set of criteria's were specified, which the testbed must fulfil to be able to measure and evaluate the performance-related drawbacks. With these criteria the hardware setup of the testbed could be specified. The next step was the definition of the needed test programs and tools, so that all drawbacks could be measured. With the resulting testbed it is possible to evaluate the security solutions in a controlled environment. All important properties of the test environment can be specified, so all results are reproducible. With some installed test tools, e.g. Ethereal it is possible to check the correctness of the measured results. After implementing the testbed, all features and the three security solutions have been verified.

This testbed is designed to quantify the performance related drawbacks of security solutions for SCTP. So, in a future work, which will be my Bachelor thesis, I will actually perform these tests. The performance drawbacks of the security solutions are caused by SCTP features which are not supported in an effective way. First of all, a set of scenarios have to be developed, to measure the effect of these SCTP features on the security solution atomically. Only if the scenarios are well defined, it is assured that the measured results are only affected by the SCTP feature responsible for the drawback and not by any other effects. The security solutions can be used in many areas, e.g. in local area networks or the internet, so the solutions have also to be evaluated for different environments. The testbed is not only able to measure effects of the security solutions in combination with SCTP, it is also possible to measure general SCTP behaviour. So another part of my Bachelor thesis related to SCTP security will be to measure the effect of different Denial of Service (DoS) attacks on SCTP endpoints. SCTP uses a cookie mechanism as a protection against blind DoS attacks. But until now, this effectiveness of this cookie mechanism has not been validated in measurements. Another interesting point which could be investigated is the possibility of new DoS because of SCTP specific characteristics, i.e. a DoS using the cookie mechanism by sending fake cookies to a server, or replaying a valid cookie. My Bachelor thesis deals with a general SCTP security investigation, and the planned measurements and investigation may reveal further security weaknesses of SCTP which can be investigated in more detail.

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

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

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

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

[8] 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

[9] 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

[10] 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

[11] 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

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

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

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

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

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

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

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

[19] SCTPLIB downloads: http://www.sctp.de/sctp-download.html

[20] Spenneberg, R.: IPsec HOWTO, http://www.ipsec-howto.org, July 2005

[21] OpenSSL downloads: http://www.openssl.org/source/
[22] R-Project description and downloads: www.r-project.org
[23] GTK/GLIB downloads: ftp://www.gtk.org/pub/gtk/v1.2/
[24] 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
[25] Jungmaier, A.: "Das Transportprotokoll SCTP – Leistungsbewertung und Optimierung eines neuen Transportprotokolls", Dissertation, August 2005

Appendix

Appendix A: General Testbed Parameters

Hardware of Computer A:

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

Hardware of Computer B:

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

Software of the peer computer:

- Suse Linux 9.1
- Kernel 2.6.5-7.151-default i686 athlon i386 GNU/Linux
- KDE 3.2
- Selections:
    - KDE complete
    - C/C++ Compiler and Tools
    - Network/Server
    - Kernel Development
    - KDE Development
    - Experienced User

Hardware of the 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

Software of the Router:

- FreeBSD 5.4-RELEASE AMD64
- Distribution sets:
    - Base
    - Developer

Appendix B: Configuration and Execution Guides

Excerpt of the router's /etc/rc.conf file

```
hostname="sctprouter.uni-due.de"
defaultrouter="132.252.152.129"

# -- lowering the security level  --
kern_securelevel="-1"
kern_securelevel_enable="YES"

# -- setting up the interfaces ---
ifconfig_sk0="inet 132.252.152.233  netmask 255.255.255.128"
ifconfig_em0="inet 10.1.1.1  netmask 255.255.255.0"
ifconfig_em3="inet 10.1.4.1  netmask 255.255.255.0"
ifconfig_em2="inet 10.1.3.1  netmask 255.255.255.0"
ifconfig_em1="inet 10.1.2.1  netmask 255.255.255.0"

# -- allow forwarding/routing
gateway_enable="YES"

# -- ipfw for dummynet --
firewall_enable="YES"
firewall_script="/etc/ipfw.rules"
```

Excerpt of the router's /boot/loader.conf file

```
dummynet_load="YES"
```

Commands to install the adapted ethereal program

```
tar -zxf ethereal-sec.tar.gz
cd ethereal-0.10.0a
./configure --with-glib-prefix=/opt/gnome
make
make install
```

(You need to start ethereal as root!)

Settings for ipfw/dummynet

The following script makes dummynet to simulate one T1 link between the interfaces em0 and em3 and a second T1 link between the interfaces em1 and em3. Additionally packets get lost with the probability of 0.07 at each link in only one way (from em0 to em2 and from em1 to em3 respectively).

```
ipfw -q -f flush                                    # Deleting all rules
```

```
ipfw add 65534 allow ip from any to any             # Allowing all packets in
principle

em0="10.1.1.0/24"                                   # Defining variables
em1="10.1.2.0/24"
em2="10.1.3.0/24"
em3="10.1.4.0/24"

ipfw add 99 prob 0.07 drop ip from $em0 to $em2 out  # Dropping with given
probability
ipfw add 100 pipe 1 ip from $em0 to $em2 out        # Directing rest into a pipe

ipfw add 199 prob 0.07 drop ip from $em1 to $em3 out # Dropping with given
probability
ipfw add 200 pipe 2 ip from $em1 to $em3 out        # Directing rest into a pipe

ipfw add 300 pipe 3 ip from $em2 to $em0 out        # Directing into a pipe
ipfw add 400 pipe 4 ip from $em3 to $em1 out        # Directing into a pipe

ipfw pipe 1 config bw 1540Kbit/s                    # Limiting bandwidth of pipes
ipfw pipe 2 config bw 1540Kbit/s
ipfw pipe 3 config bw 1540Kbit/s
ipfw pipe 4 config bw 1540Kbit/s
```

Getting started with SCTP source

First we need to check out the sources from the SVN server:

```
svn co svn+ssh://<accountname>@svn.iem.uni-due.de:/home/svn/sctp/trunk/sctplib1
svn co svn+ssh://<accountname>@svn.iem.uni-due.de:/home/svn/sctp/trunk/socketapi
```

Alternatively we can download the archives and unpack them:

```
tar -zxvf sctplib-1.0.4.tar.gz
tar -zxvf socketapi-1.7.0.tar.gz
```

Now we have to compile and install the source:

```
./configure --with-glib-prefix=/opt/gnome
make
make install
```

By using the following commands simple SCTP communication can be checked with the programs which come with the SCTP sources:

Server:
```
sctplib1/sctplib/programs/echo_server
```

Client:
```
sctplib1/sctplib/programs/terminal -v -d 10.1.1.2
```

IPsec setup

This is an example of the *psk.txt* file, which holds the pre-shared keys, on the server computer:

```
10.1.3.2        racoon_password
10.1.4.2        racoon_password
```

The following lines show the *ipsec.conf* file on server:

```
#!/usr/sbin/setkey -f

# Configuration for 10.1.1.2

# Flush the SAD and SPD
flush;
spdflush;

spdadd 10.1.1.2 10.1.3.2 any -P out ipsec
           esp/transport//require;
spdadd 10.1.3.2 10.1.1.2 any -P in ipsec
           esp/transport//require;

spdadd 10.1.2.2 10.1.4.2 any -P out ipsec
           esp/transport//require;

spdadd 10.1.4.2 10.1.2.2 any -P in ipsec
           esp/transport//require;
```

The following lines show the *racoon.conf* file on server:

```
path pre_shared_key "/etc/psk.txt";

remote 10.1.3.2
{
        exchange_mode main;
        proposal {
                encryption_algorithm 3des;
                hash_algorithm md5;
                authentication_method pre_shared_key;
                dh_group modp1024;
        }
}

sainfo address 10.1.1.2 any address 10.1.3.2 any
{
        pfs_group modp768;
        encryption_algorithm 3des;
        authentication_algorithm hmac_md5;
        compression_algorithm deflate;
}
remote 10.1.4.2
{
        exchange_mode main;
        proposal {
                encryption_algorithm 3des;
                hash_algorithm md5;
                authentication_method pre_shared_key;
                dh_group modp1024;
        }
}

sainfo address 10.1.2.2 any address 10.1.4.2 any
{
```

```
        pfs_group modp768;
        encryption_algorithm 3des;
        authentication_algorithm hmac_md5;
        compression_algorithm deflate;
}
```

Commands to apply the configuration of the files:

```
setkey -f /etc/ipsec.conf
racoon -f /etc/racoon.conf
```

Adapting files for openssl using SCTP (by Unurkhaan)

Makefile file:

```
EX_LIBS= -L/opt/gnome/lib
/home/attacker/openssl_tls/openssl-0.9.7c/libssl.a
/home/attacker/openssl_tls/openssl-0.9.7c/libcrypto.a
/home/attacker/ssctp/socketapi/socketapi/.libs/libsctpsocket.a
/home/attacker/ssctp/sctplib1/sctplib/sctp/.libs/libsctp.a
/home/attacker/openssl_tls/open ssl0.9.7c/libcrypto.a -ldl -lglib (←)
-lpthread -lstdc++

PERL= /usr/bin/perl5.8.3
```

bss_sock.c file:

```
#include <ext_socket.h>   or   #include <sys/ext_socket.h>
```

Also curly brace is set wrong:
Move brace in line 156 to line 203.

ssl.h file:

```
#include <ext_socket.h>   or   #include <sys/ext_socket.h>
```

Adapting files for S-SCTP (by Unurkhaan)

A few entries in the file *sctplib/programs/Makefile* of the S-SCTP verstion (by Unurkhaan) must be changed from `/usr/local/ssl/lib/` to `/usr/lib/` .

For installing the socket-API I copied the following files to the specified location:

ssctp_messages.h	=>	*/usr/local/include*
messages.h	=>	*/usr/local/include*
/opt/gnome/include/glib-1.2/glib.h	=>	*/usr/local/include*
/opt/gnome/include/glib-1.2/gmodule.h	=>	*/usr/local/include*
/opt/gnome/lib/glib/include/glibconfig.h	=>	*/usr/local/include*

Adapted lines *Makefile* file of *test-ssctp-serv.cpp*

```
COPY-S-SAPI=/home/server/ssctp/socketapi/socketapi
SSLDIR=-I /usr/local/ssl/include/openssl -L /usr/local/ssl/lib
OPENSSL=/home/server/openssl-0.9.7c
```

Example of using test-ssctp-serv and test-ssctp-cli

To use the programs following command lines are possible.
Server:

```
./test-ssctp-serv 7 1 100 1 10.1.1.2 1 300 1 0
```

- local (server) port: 7
- protocol: 1 (0 => TCP, 1 => SCTP)
- frame size: 100
- stream numbers: 1
- local (server) address: 10.1.1.2
- out of the blue (OOTB) handling: 1
- count (retries): 60
- …

Client:

```
./test-ssctp-cli 7 1 100 1 10.1.3.2 10.1.1.2 1 60 1 0
```

- destination (server) port: 7
- protocol: 1 (0 => TCP, 1 => SCTP)
- frame size: 100
- stream numbers: 1
- client address: 10.1.3.2
- server address: 10.1.1.2
- out of the blue (OOTB) handling: 1
- count (retries): 60
- …

Perftool usage

Starting the responder using SCTP, listening at all available IP interfaces at port 4711:

```
./perftool -sctp -responder=4711
```

Starting the transaction:

```
./perftool -sctp -port=4711 -hosts=remote.hosts -scenario=transaction
-chunksize=100 -chunksize=1000 -chunksize=5000 -chunksize=10000 -chunkrate=10
-startup=1 -runtime=3 -runs=3 -connecttimeout=3 -vector=vector1.bz2 -scalar=scalar1
```

Options:

`-sctp` | `-tcp`	using SCTP or TCP
`-port=<port-number>`	port number of the peers
`-hosts=<filename>`	file with list of peer hosts (the responders)
`-scenario= {transaction|throughput}`	sets scenario: transaction runs with determined loops; throughput sends as fast as possible
`-chunksize=<chunk_size>`	sets the chunk size in byte for each run
`-chunkrate=<chunks_per_sec>`	sets the chunk rate of all runs
`-startup=<sec>`	sets startup time before measuring starts
`-runtime=<sec>`	sets duration of all runs
`-runs=<number_runs>`	sets iteration of runs
`-connectiontimeout=<sec>`	sets timeout interval
`-vector=<filename>`	sets output filename for bz2-compressed vector
`-scalar=<dir_name>`	sets directory name for statistical scalar files

Continuing with entering the following command creates the summary:

```
./createsummary <scalar1/summary.cfg
```

With the provided *R* scripts *plotter.R* and *plot-test1.R* the data can easily be used with the *R* program. The plot-test1.R file needs to be adapted to the data which need to be displayed with the graphs. Starting the *R* tool with `R –vanilla` and entering the command `source("<plot-test1.R")` starts the gnuplot window and displays the graph. Note: *R* must not be started by user *root*.

Glossary

CRC	Cyclic Redundancy Check
DoS	Denial of Service
HTTP	HyperText Transfer Protocol
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force
IEM	Institute for Experimental Mathematics
IP	Internet Protocol
ISAKMP	Internet Security Association and Key Management Protocol
MAC	Message Authentication Code
MTU	Maximum Transfer Unit
NIC	Network Interface Card
OOTB	Out Of The Blue (Packet)
OSI	Open Systems Interconnection
PMTU	Path MTU
RFC	Request For Comment
RTO	Retransmission TimeOut
RTT	Round Trip Time
SI	Stream Identifier
SIGRANS	Signalling Transport
SSL	Secure Socket Layer
SSN	Stream Sequence Number
TCB	Transmission Control Block
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TSN	Transmission Sequence Number
UDP	User Datagram Protocol
ULP	Upper Layer Protocol