SSH over SCTP — Optimizing a Multi-Channel Protocol by Adapting It to SCTP

Robin Seggelmann, Michael Tüxen, Erwin P. Rathgeb

Abstract—Secure Shell (SSH) is a multi-channel security protocol running over the Transmission Control Protocol (TCP), which offers channels for several services over a secured connection, such as remote shells and connection forwarding.

In this paper we introduce a method for using SSH over the Stream Control Transmission Protocol (SCTP), a transport protocol supporting multi-homing and multi-streaming. We examine benefits of this adaptation, which can be made available to generic applications with SSH’s connection forwarding without further changes. The results also apply to other application protocols supporting a concept similar to channels.

I. INTRODUCTION

WITH the wide deployment of the Internet, security has become a more and more important issue, and so the protection of services with encryption and authentication is common now. A versatile security protocol is Secure Shell (SSH) [1], which offers a variety of secured services, such as remote shells, data transfer or connection forwarding. SSH uses the Transmission Control Protocol (TCP) as its transport protocol.

The Stream Control Transmission Protocol (SCTP) [2] is a transport protocol, which supports multiple IP-addresses at each endpoint and the multiplexing of multiple streams within a single connection. Using it for multi-channel application protocols, like SSH, is a promising approach to optimize the performance. The additional benefit is that these performance improvements can be made available with SSH’s connection forwarding for basically every TCP based application protocol without further changes. In the following, we will discuss the necessary adaptations and analyze the performance improvements due to SCTP features such as multi-homing and multi-streaming.

This paper is structured as follows: In Sections II and III the protocols SSH and SCTP are introduced. The related work is described in Section IV. The concept of using SSH over SCTP is discussed in Section V. In Section VI a prototype implementation is described, which is used to evaluate the expected benefits in Section VII.

II. SECURE SHELL

Secure Shell (SSH) is a security protocol providing bi-directional channels for various services over a single secure connection. The secure connection is established by using the SSH Transport Layer [3]. After performing the key exchange to negotiate the security parameters, as shown in Figure 1, it provides the secure transport for the two other SSH subprotocols: The SSH Authentication protocol and the SSH Connection Protocol.

The SSH Authentication [4] protocol is used by the client to request the permission to use a specific service. It provides a username, and the server offers acceptable authentication methods, such as a password or public-key. Upon successful authentication, the client is allowed to request the service, which is done with the SSH Connection [5] protocol. This protocol provides channels for each offered service, which can be opened and closed arbitrarily within the single secure connection. The structure of SSH and its subprotocols is illustrated in Figure 2.

A common service is the port or connection forwarding for TCP, as illustrated in Figure 3. An SSH connection is set up between two hosts, the SSH endpoints, and one of them is configured to accept all TCP connections on a specific port. The data received from connections accepted on this port will be forwarded over the SSH connection to the other SSH endpoint, which will establish a new TCP connection to deliver the forwarded data to an also previously configured destination host. Every accepted incoming TCP connection triggers the opening of a forwarding channel within the SSH connection, and the establishment of the new TCP connection to the configured destination by the other SSH endpoint.

---

R. Seggelmann and M. Tüxen are with Münster University of Applied Sciences, Stegerwaldstrasse 39, 48565 Steinfurt, Germany. Email: {seggelmann, tuexen}@fh-muenster.de

E. P. Rathgeb is with University of Duisburg-Essen, Ellerntstrasse 29, 45326 Essen, Germany. Email: erwin.rathgeb@iem.uni-due.de
The payload of the initial TCP connection is received by the first SSH endpoint, sent over the forwarding channel to the other SSH endpoint, which sends it over the new TCP connection to the destination and vice versa. This service can be used to secure an otherwise unsecured connection, since the SSH connection is encrypted, or to tunnel a connection to a protected host, since SSH requires authentication.

SSH maintains a flow control for every data transmission channel, realized with a window indicating the amount of data that is allowed to be sent on this channel. The size of the window is initially announced during the channel opening and then continuously adjusted. This allows to slow down the transfer rate of a forwarded sender, in case the receiver cannot deliver the incoming data to the destination quickly enough or not at all.

III. STREAM CONTROL TRANSMISSION PROTOCOL

The Stream Control Transmission Protocol (SCTP) [2] is a reliable and message-oriented transport protocol supporting multiple streams over a single connection, called association. Streams are unidirectional channels to separate logically independent data. SCTP only preserves the order of the messages within a single stream to mitigate an effect known as head-of-line blocking. If a message is lost, the subsequent messages have to be delayed until the lost one is retransmitted to preserve the order. Since only messages of the same stream have to be delayed, the number of affected messages after a loss is reduced. Figure 4 shows the protocol structure with multi-streaming.

For increased reliability, multi-homing is supported, so multiple addresses can be used for a single association. The additional addresses are negotiated during the connection establishment and can be used as a fallback in case of failure. SCTP has an extensible protocol design, which allows adding new features without changing the base protocol. An SCTP packet starts with a Common Header, which is followed by one or multiple chunks. These chunks can contain control information, such as Selective Acknowledgements (SACK), or user data. New chunk types can be defined by extensions. The Concurrent Multipath Transfer (CMT) extension, see [6] and [7], allows to increase the available bandwidth by using multiple paths through the network simultaneously. Addresses used for an association can be added and removed for an already established association with the ADD-IP [8] extension. A detailed introduction to the protocol can be found in [9].

IV. RELATED WORK

Modifications of the SSH protocol to extend its features or improve its performance have already been proposed. In [10] an extension is suggested to add mobility to SSH connections. The mobile host monitors its interfaces, and as soon as an address change is detected, the connection to the SSH server is re-established. The SSH packet header is extended with a session identifier, which is used by the server to look up the existing session and continue it with the new address.

In [11] a high performance modification for SSH is described. The behavior of the flow control is adjusted to allow a higher throughput for modern networks, which has already been considered in recent OpenSSH releases. Furthermore, multi-threading support for the encryption and decryption is...
proposed to make use of modern multi-core CPUs and avoid a throughput limitation because of an overloaded single core.

A generic approach to extend TCP based applications with multi-homing capabilities, called pTCP, has been introduced in [12]. Unfortunately, no implementations are available yet.

V. SSH OVER SCTP

Although TCP is commonly used, the specification of the SSH Transport Layer only requires a transport protocol, which protects against transmission errors (lost messages are repeated) [3] and preserves the order of the messages. SSH uses multiple channels to realize its services, but they are multiplexed for the secured connection provided by the SSH Transport Layer. As a result, the protocol expects a single ordered transport channel for its SSH Transport Layer from the transport protocol. This corresponds to a single-homed SCTP association with a single stream, which behaves just like TCP.

A. Usage of Multi-Homing

The immediate benefit of using SCTP is the multi-homing support that increases the reliability or bandwidth of SSH connections. The necessary modifications are limited to changing the sockets to SCTP and extending the configuration, to allow specifying multiple addresses for multi-homing support. With its message-orientation, every SSH packet will be sent in an SCTP message. Although this reveals the message boundaries, that is not a security issue. Message boundaries are visible for interactive remote shells anyway, due to the disabled Nagle’s algorithm. Every input is sent immediately, instead of awaiting enough data to fill a packet. And for bulk transfer, always the maximum message size will be used, which is 32 KBytes in the current OpenSSH implementation. The ADD-IP extension for SCTP [8] that allows adding and removing addresses of already established connections, can even be used to realize mobility as described in [13].

B. Usage of Multi-Streaming

The multi-streaming feature of SCTP is a concept similar to the channels of an SSH connection, both are used to separate logically independent data. The SCTP streams are not only used to mitigate the impact of head-of-line blocking on lossy links, with specialized stream schedulers a fair distribution of the available bandwidth or the prioritization of specific streams, like those carrying interactive remote shells, can also be achieved [14].

Although the message order across multiple SSH channels is arbitrary, they are multiplexed before being carried by the SSH Transport Layer, which prevents any further reordering. Since SCTP does not preserve the order of messages across multiple streams, the reordering of messages of multiple channels must be possible at any time to allow the mapping. Therefore, the SSH Transport Layer needs to be adapted to tolerate reordering across multiple channels, while still requiring the messages within a single channel and of the other subprotocols to stay in order.

SSH uses a sequence number for each message for the calculation of the Message Authentication Code (MAC) to ensure integrity. This sequence number is not transmitted; both endpoints keep track of it instead. If a message arrives out of order, the assumed sequence number is different, and the verification of the MAC and therefore the entire connection fails. To prevent this, the message header of the SSH Transport Layer has to be extended with the sequence number to transfer it explicitly, like it is done with DTLS or the Authentication Header of IPsec. Contrary to these protocols, the entire SSH header is encrypted, so the sequence number will not be revealed. Additionally, the sequence number has to be extended to differentiate between messages that have to arrive in order or that may be reordered. As a possible solution it is suggested to use multiple sequence numbers, one for the SSH Transport Layer, one representing the sub-protocol or channel, and one for the messages within a channel. The first sequence number is increased for every SSH Transport Layer message, such as key exchange messages. Every time a different sub-protocol is used, the first sequence number is also increased but stays the same for all messages of the sub-protocol, and the second sequence number is used for the messages of the sub-protocol. In case of channels, every channel has its own second sequence number, and the third number is used for each of its messages. The order can be verified as follows:

- If only the first sequence number is set, they have to be in order.
- If the first and second sequence numbers are set, the second number has to be in order for each first number.
- If all three sequence numbers are set, the third number has to be in order for each second number.

Each sequence number has to have a size of 32 bit, like the original sequence number, so it is also ensured that a wrap around and thus a number reuse still only occurs after at least $2^{32}$ messages.

A similar issue is caused by cipher suites which are depending on the message order, for instance because a counter (counter mode) or data from the previous message (cipher-block chaining) is used as the Initialization Vector (IV) for the encryption. Generating random IVs, which are then prepended to the SSH Transport Layer message, can solve the dependency. The Transport Layer message format with the extensions is illustrated in Figure 5.

The SSH connection is established with a key exchange of the SSH Transport Layer. This negotiation can be repeated at any time to refresh the key material of an existing connection. No other messages are allowed during the negotiation, and the message sequence of the key exchange has to be preserved,
which can be done by sending all of its messages on the same SCTP stream. When mapping channels onto streams, however, channel messages on other streams may arrive during the exchange if they arrive too late or too early. To prevent this, the message order has to be enforced across all streams while a key exchange is performed. This requires to drain the transfer on all streams except the one the key exchange is performed on. Hence, before sending the Key Exchange Init and before resuming the regular transmission after the New Keys message (compare Figure 1), the reception of all previous messages has to be awaited before continuing. The same is necessary before shutting down the SSH connection. SCTP supports a SENDER_DRY event to notify the upper layer that all outstanding messages have been delivered successfully, which can be used to realize this.

The packet size of SSH channel messages should be limited to the Maximum Transmission Unit (MTU), the maximum possible message size. The default packet size of SSH may be larger and therefore messages have to be fragmented by SCTP, which limits the effect of stream scheduling, because all fragments have to be transmitted before the scheduler can select the next message.

The mapping of channels onto streams, however, causes a confidentiality issue, which has to be considered. The header of SCTP data messages is not encrypted; therefore, an attacker knows how many streams are used and thus can estimate the number of channels. If more channels are used than streams are possible, multiple channels have to be mapped onto each stream and the disclosure of the exact channel number is avoided. But since SCTP supports 2^16 streams, this should occur rarely and the exact number of channels may be revealed.

Using encryption for the transport protocol to protect the header information can solve this issue. Since there is no such solution available, we propose the introduction of a new extension called the SCTP Encryption Chunk. With the Encryption Chunk extension, each peer can, during the handshake for the connection establishment, announce which encryption algorithms it supports and which types of chunks it will only accept encrypted. The relevant chunks will then, rather than being sent as plain text, be encrypted, and the result is sent in a new chunk type, the encryption chunk. The receiver decrypts the content of the encryption chunk and can process the resulting original chunks as usual. This process is shown in Figure 6. The necessary shared secret can be derived from the secret that is negotiated for the SSH connection.

VI. IMPLEMENTATION

The implementation of SSH over SCTP can be separated into two parts. The first is just using SCTP as the transport protocol without modifications of the SSH protocol. The second, and more extensive, adds mapping of channels onto streams, which requires adaptations of the SSH Transport Layer. We have developed a prototype implementation based on the common open source tool OpenSSH [15] for the evaluation of the proposed adaptations.

A. Adding Support for Multi-Homing

To use SCTP as the transport protocol, the socket creation has to be modified. For backward compatibility, the use of SCTP should be additional to TCP and configurable. OpenSSH uses multiple sockets, one for each listening address, so adding SCTP sockets for the same addresses is rather straightforward. When using multi-homing, however, multiple addresses have to be bound to a single socket. Therefore, the configuration has to be extended to allow the differentiation between multiple addresses bound to separate sockets or to a single one. The handling of TCP specific socket options has to be changed appropriately, if an SCTP socket is used. With these comparably small modifications, SSH can already be used over SCTP including its multi-homing features.

B. Adding Support for Multi-Streaming

The mapping of channels onto streams requires to track a socket’s protocol if TCP and SCTP should be supported concurrently. In case SCTP is used, the appropriate stream has to be set according to the current channel, and the modified SSH Transport Layer has to be used. The latter includes the different handling of sequence numbers, as well as the generation of a random IV for the encryption instead of using data of the previous message to allow the reordering of channel messages. The message order has to be enforced during key exchanges, so before sending a Key Exchange Init and before resuming normal transmission after a New Keys message, the reception of all data still in flight has to be awaited. The same needs to be done before shutting down the SSH connection. So instead of sending the Key Exchange Init message or the next channel data, the SENDER_DRY event has to be activated with a socket option. When the SCTP stack sends a notification that no data is outstanding anymore, the actual action can be performed. Finally, the maximum packet size has to be limited to avoid fragmentation, which is set in the channel opening messages. After these modifications, SSH can already benefit from being less susceptible to head-of-line blocking. A specialized stream scheduler, like prioritization, can be activated with a single socket option.

VII. EVALUATION

To measure the benefits of the proposed optimizations, we implemented SSH over SCTP based on OpenSSH, and also an OMNeT++ simulation [16] behaving exactly the same. The simulation allows to evaluate the maximum improvements that can be expected in a controlled environment. Measurements with real systems, on the other hand, allow to assess the influences of factors like hardware limitations on the expected results.

The real system measurements have been done with two FreeBSD 9.0 (Beta2) hosts, each having two quad core 2.4
GHz Xeon CPUs. The hosts are directly connected via four gigabit links as shown in Figure 7. The bandwidth, delay, and packet loss rate of each link can be configured with Dummyynet [17] on both hosts. An SSH connection with local TCP forwarding is set up between the hosts. For simplicity, and to avoid additional delays, the senders and the SSH client were run on one host, and the SSH server and the receiver on the other. Because each host had eight CPU cores and all applications were single-threaded, this did not affect the performance. The delays and achieved throughput of each forwarded connection were measured.

Multiple configurations have been used for the series of measurements to cover the scenarios in which SCTP behaves different to TCP. Each measurement has been repeated 25 times to calculate average values and 95% confidence intervals.

A. Reliability with Multi-Homing

SCTP offers multi-homing to increase the reliability by using multiple network links for a single association. To evaluate the benefit of this feature, the time necessary to transfer 100 MB of data over a forward saturated TCP connection is measured, while network failures are simulated during the transfer. Two links are used and their bandwidth is limited to 100 MBit/s to avoid any side effects from a maximum load of the CPUs. Five seconds after the transmission started, the failure of one network link is simulated by configuring the packet loss rate to 100%. The duration of the outage is increased during the measurement. While SSH over TCP can only use a single link, SSH over SCTP is dual homed, so a fallback link is available.

The results of the simulation of this scenario are shown in Figure 8. Without any network failures, the transmission lasted about 9 seconds with both TCP and SCTP. If there was an outage, TCP retransmitted the last packet until the link was available again, so the transfer time was prolonged by at least the duration of the outage. However, since TCP’s timers are increased exponentially, the prolongation was not linear, because even when the network was already available again, the time until the next retransmission still had to be awaited. This caused the graph to increase in steps, because as long as the duration of the outage did not require an additional retransmission, the time until the data transfer resumed stayed the same. In the simulation the initial RTO was 200 ms and doubled for every retransmission, as stated by the specification. This can be described with \(0.2 * 2^{n-1}\) s, where \(n\) is the number of retransmissions. Therefore, the overall time necessary until retransmission \(n\) can be described with Equation 1.

\[
\sum_{x=1}^{n} 0.2 * 2^{x-1} s. \tag{1}
\]

So if the link outage lasts 4 seconds for example, the first packet arrives after 5 retransmissions or 6.2 seconds. After 60 seconds of network outage, the SSH connection over TCP was terminated because of too many unsuccessful retransmissions.

SCTP was measured with its Quick Failover extension [18] activated, so the failure of the primary network link was assumed after a single Retransmission Timeout (RTO). For SCTP, this is by default one second compared to only 200 ms for TCP. Therefore, the time to complete the transfer was only prolonged by the time necessary to detect the failure and change to a fallback link, which was done within a second, no matter how long the primary link was actually unavailable.
The same measurement has also been done on real systems and the results are shown in Figure 9. While the behavior of SCTP is exactly the same as in the simulation, the time necessary to complete the transfer with TCP was slightly better. This was because FreeBSD uses a different retransmission strategy to lower the time between the retransmissions. By default, the initial value is 30 ms and is doubled for every retransmission, but in addition 200 ms are added, which can be described as \(0.2 + (0.03 \times 2^n - 1)\) s. The overall time necessary until retransmission \(n\) can be described with Equation 2.

\[
\sum_{x=1}^{n} 0.2 + (0.03 \times 2^{x-1}) \text{ s.} \quad (2)
\]

In Table I this approach is compared to that of the specification. So if the link outage lasts again 4 seconds, with the approach of FreeBSD the transfer resumes after 8 retransmissions or 5.31 seconds. Hence, FreeBSD is more efficient in resuming the transfer after an outage, but uses significantly more retransmissions, which can worsen the situation in case the losses are caused by a network congestion.

B. Throughput with Multi-Homing and CMT

The second possible benefit of multi-homing is to increase the throughput by using multiple network links simultaneously with the CMT extension [19] for SCTP. Only the basic features of the extension were used, since maintaining fairness was not an issue in this scenario. To evaluate how the throughput increases with multiple links, a saturated TCP connection was forwarded over an SSH connection using multiple links with SCTP and a single one with TCP. The configured bandwidth of the links was increased from 1 to 200 MBit/s during the measurement and was always the same for all links.

The simulation results in Figure 10 show that as TCP can only make use of a single link, the achieved bandwidth increases linearly with the configured bandwidth. This is identical to SCTP with a single link, but with multiple links, SCTP uses all available links with CMT simultaneously, so the throughput is the configured bandwidth times the number of links used, up to almost 800 MBit/s with four links at 200 Mbit/s.

The comparison with the real systems, displayed in Figure 11, basically shows the same results, although the hardware limitation becomes visible. The possible throughputs only increase until the CPU cannot perform the encryption and decryption fast enough. This limit is reached at about 550 MBit/s, slightly more without CMT and slightly less with CMT. This is because CMT requires more computing power and the implementation is still experimental, so optimizations have not been done yet.

The measurement shows that until a hardware limitation is reached, which can be mitigated by using multiple CPU cores for the cryptographic calculations of SSH as described in [11], the possible throughput can be increased with additional network links, which is especially effective on low bandwidth links, such as Internet connections.

C. Delay with Mapping Channels onto Streams

By mapping the channels of SSH onto SCTP’s streams, the benefit of multi-streaming is made available to SSH, which is the mitigation of head-of-line blocking. To evaluate this, four saturated TCP connections are forwarded over SSH with a single link, which has a delay of 25 ms and a packet loss rate of 1% configured. These values were chosen to represent a 3G Internet link with High Speed Downlink Packet Access (HSDPA) [20], which is common nowadays. The delay of each message of the forwarded connections, from being sent to its reception, is measured. Figure 12 shows the distribution function of the message delays for the simulation.

With the configured delay and packet loss rate, 1% of the packets are expected to be lost and therefore delayed until their retransmission, while the others should arrive after the link latency of 25 ms. With TCP, however, almost 15% of the messages were delayed, because the order of the messages had to be preserved. Therefore, already received messages could not be delivered until previous ones had been retransmitted. The steps of the graph represent the retransmissions performed after each Round-Trip Time (RTT).

This issue is mitigated by SCTP’s multi-streaming, which only requires the delay of messages of the same stream in case of a loss. Four connections were forwarded in this scenario, so four streams were used. This resulted in a delay of only about 5% of the messages on the same link.
TABLE I
RETRANSMISSION TIMINGS

<table>
<thead>
<tr>
<th>Retransmission</th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
<th>#4</th>
<th>#5</th>
<th>#6</th>
<th>#7</th>
<th>#8</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP Specification</td>
<td>0.200s</td>
<td>0.400s</td>
<td>0.800s</td>
<td>1.600s</td>
<td>3.200s</td>
<td>6.400s</td>
<td>12.800s</td>
<td>25.600s</td>
</tr>
<tr>
<td>TCP in FreeBSD</td>
<td>0.230s</td>
<td>0.260s</td>
<td>0.320s</td>
<td>0.440s</td>
<td>0.680s</td>
<td>1.160s</td>
<td>2.220s</td>
<td>4.040s</td>
</tr>
</tbody>
</table>

Fig. 12. Simulation of Message Delays

These results were confirmed with the real systems in Figure 13. The measurement therefore shows that SCTP can improve the message delay on lossy links, which is a benefit for time critical applications, such as interactive remote shells or real-time monitoring traffic.

D. Delay with different Stream Schedulers

Another benefit of multi-streaming is the possibility to use specialized stream schedulers. SCTP supports the selection of a specific scheduler for each association and as described in [14], all standard scheduling algorithms can be used. A possible application is to prioritize interactive and therefore time critical data, like remote shells or X11 forwarding. Such a scenario is measured by forwarding four TCP connections, three saturated and one only sending a message every 500 ms. This simulates file transfers while a remote shell is used with the same SSH connection. A single link is configured to 10 MBit/s without a delay and therefore forms a bottleneck for the SSH connection. Hence, all forwarded connections have to share the reduced bandwidth. The SCTP stream schedulers that are compared to TCP are First-Come, First-Served (FCFS, default on Linux and Solaris), Round-Robin (default on FreeBSD) and a priority scheduler. The message delay of the unsaturated “interactive” connection is measured.

Figure 14 illustrates the results of the simulation. In this scenario all messages were delayed because of the bottleneck, so they remained in the send buffer until they could be transmitted. With FCFS scheduling, all messages were processed in the order in which they arrived from the application. The same was true for TCP, which does not support multi-streaming at all. Because the majority of the messages on the SSH connection belonged to the saturated connections, which filled up the send buffer, the interactive messages had to wait a considerably long time until they could be sent, resulting in the large delay of about 200 ms for TCP and SCTP with FCFS scheduling.

A round-robin scheduler on the other hand cycles through all available streams when choosing the next message that will be sent. In this case there were four streams, so the stream with the interactive messages was chosen every fourth time. Whenever an interactive message was available, there were never more than three messages that were sent first, so the delay was about 110 ms and so significantly less than with FCFS scheduling or with TCP. With priority scheduling, a higher priority can be assigned to the streams with interactive service channels. Even if there are many other messages already in the send buffer, a high priority message will always be sent first. This results in the smallest possible delay, which is especially effective when many streams are used, or the transmission of a single message takes a long time because...
of large delays or a very low bandwidth. In this case only four streams have been used, so the difference to round-robin scheduling was still very small.

The same behavior could be observed with the real systems with only small deviations, as shown in Figure 15.

VIII. CONCLUSION AND OUTLOOK

In this paper we suggested to use SCTP as the transport protocol for the multi-channel SSH protocol to improve its performance with SCTP’s multi-homing and multi-streaming and also make these features available to generic applications with SSH’s connection forwarding. Simply using SCTP improves the reliability and the possible throughput by using multiple network links with the multi-homing support. Mapping the channels onto streams allows to use specialized stream schedulers and reduces the number of delayed messages on lossy links. The adaption of the SSH Transport Layer is necessary to handle the new reordering constraints. To avoid security issues, a new SCTP extension called SCTP Encryption Chunk has been proposed.

In the future, we will investigate the benefits of a specialized congestion control for SCTP with multi-channel protocols. In addition, we will contribute the SSH modifications and the SCTP Encryption Chunk to the Internet Engineering Task Force (IETF) for standardization and our implementation to the OpenSSH maintainers.

REFERENCES


Erwin P. Rathgeb (rathgeb@iwm.uni-due.de) holds the Alfred Krupp von Bohlen and Halbach-Chair for Computer Networking Technology at the Institute for Experimental Mathematics, University of Duisburg-Essen, Germany. He is the author of a book on ATM and has published numerous research papers. He is a member of GI, IFIP, and ITG, where he is chairman of the expert group on network security. In addition to security, his current research interests include concepts and protocols for future networks. He has been involved with SCTP since the initial standardization steps.

Robin Seggelmann (seggelmann@fh-muenster.de) studied Computer Science at the Martin Luther University of Applied Sciences. He graduated with a Masters Degree in 2008, and continued working as a research assistant on the optimization and security of SCTP, as well as innovative applications for DTLS. This also involved the authoring of standardization documents for the IETF. After receiving his Dr. rer. nat. at the University of Duisburg-Essen in 2012, he started working on enterprise solutions for data management and system integration at T-Systems International GmbH.

Michael Tüxen (tuxen@fh-muenster.de) studied mathematics at the University of Göttingen and received his Dipl. Math. degree in 1993 and Dr. rer. nat. degree in 1996. In 1997 he joined the Systems Engineering group of ICM WN CS of Siemens AG, Munich. Since 2003 he is a professor in the Department of Electrical Engineering and Computer Science, Münster University of Applied Sciences. In the IETF he is working on transport related issues and coauthored several RFCs. His research interests include innovative transport protocols, especially SCTP, IP-based networks, and highly available systems.