Alternative Handshake Mechanism for the Stream Control Transmission Protocol

Felix Weinrank, Irene Rüngeler, Michael Tüxen
Münster University of Applied Sciences
Department of Electrical Engineering and Computer Science
Stegerwaldstrasse 39
48565 Steinfurt, Germany
{weinrank,ruengeler,tuexen}@fh-muenster.de

Erwin P. Rathgeb
University of Duisburg-Essen
Faculty of Business Administration and Economics
Computer Networking Technology Group
Schützenbahn 70
45127 Essen, Germany
erwin.rathgeb@iem.uni-due.de

Abstract—**The Stream Control Transmission Protocol (SCTP) is a message and connection oriented transport protocol using a cookie based four-way handshake for connection establishment. The intention of the four-way handshake is to provide a robust protection against flooding attacks, like TCP SYN-flooding. Although it protects the memory resources on the server side, current implementations require a quite large amount of CPU resources. For some of SCTP's use-cases, like the underlying transport protocol for Web Real-Time Communication (WebRTC) Data-Channels, SCTP's cookie protection is unnecessary and its four-way handshake delays the connection setup unnecessarily.**

We have developed an alternative handshake method which offers a more lightweight cookie exchange and a zero round-trip time (RTT) connection setup capability while still being fully backwards compatible with the existing handshake procedure. We describe the alternative handshake procedure and its advantages in common use cases like short living connections and WebRTC Data-Channels. In addition, we evaluate the alternative handshake's benefit to the regular handshake with measurements using our FreeBSD kernel implementation.

I. Motivation

Reducing the connection setup time has become a more and more important topic in the design of network protocols. With TCP Fast Open [1], TLS 1.3 [2] and QUIC [3], many popular protocols provide specific mechanisms to reduce the connection setup time. This development is driven by the perception that connection setup time has replaced insufficient bandwidth in the context of improving the user experience.

The Stream Control Transmission Protocol (SCTP) [4] is a reliable, connection oriented transport protocol using a cookie based four-way handshake to provide protection against INIT-flooding attacks, resulting in one round trip before the client can send the first data and two round-trips until the connection establishment is completed. Even though originally designed for the transmission of small signaling messages in the Signaling System No. 7 (SS7), SCTP's use cases have been extended over the last years, for example by Web Real-Time Communication (WebRTC) where SCTP is used as the underlying transport protocol for Data-Channels [5].

We have analyzed the regular SCTP handshake in multiple scenarios and by using the FreeBSD and Linux SCTP stacks. Previous work [6],[7],[8],[9] and our evaluation shows that the handshake procedure is a resource consuming operation and vulnerable to byte amplification attacks, which is covered by the measurement and evaluation section in detail.

ISBN 978-3-903176-08-9 © 2018 IFIP

In case of WebRTC Data-Channels, SCTP's four-way handshake is redundant and adds an unnecessary delay because the SCTP communication is encapsulated within the Datagram Transport Layer Security (DTLS) [10] protocol which already authenticates the peer during its handshake.

Therefore, we developed an alternative handshake procedure focused on a more lightweight and flexible method which addresses the requirements and approaches of today's transport protocols. Our main goal was to reduce the time to set up a transport connection, while still being fully backwards compatible with the regular handshake procedure defined in RFC4960 [4].

Before we introduce our approach of the alternative handshake in detail, we explain the regular handshake and show its drawbacks with respect to setup time and resource consumption. The measurement and evaluation section points out the benefits of the new extension by running tests and benchmarks on real hardware with our FreeBSD kernel implementation. In the next section we describe the interaction with existing SCTP extensions and how the new extensions have been implemented and integrated in the existing socket API. The last section summarizes this paper and gives an outlook on further research activities.

II. Regular Handshake

An SCTP association between a server and a client is established by a four-way handshake as shown in Figure 1. The client sends an SCTP message containing an INIT chunk to an SCTP server to initialize the association setup. The INIT chunk, as shown in Figure 2, consists of several parameters carrying a variety of information. Some of the parameters are mandatory in every INIT and INIT-ACK chunk, while other parameters are optional.

The mandatory parameters include the *Initiate Tag*, *Advertised Receiver Window*, number of incoming and outgoing streams and the *Initial Transmission Sequence Number* (TSN). The optional parameters provide a flexible way to store arbitrary information in those chunks, for example a list

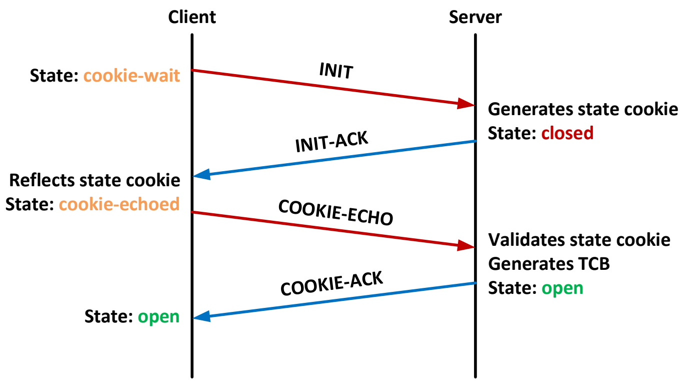

Fig. 1: Regular four-way handshake

of available IP address candidates and additional supported extensions.

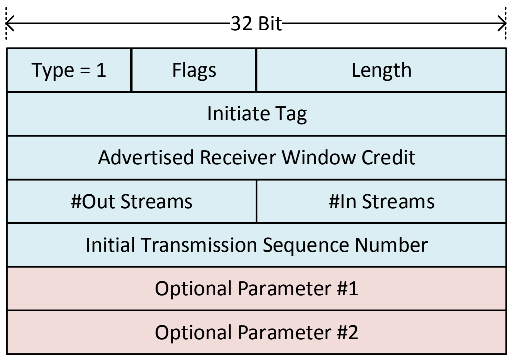

Fig. 2: INIT chunk structure

When receiving an SCTP message containing an INIT chunk, the server responds with an INIT-ACK chunk which consists of the same mandatory parameters as the INIT chunk and an additional *State Cookie* parameter. The state cookie contains all information the server needs to create the *Transmission Control Block* (TCB) and establish the association. It is digitally signed by the server to ensure its validity and neither the format nor the size of the cookie is specified by RFC4960, but it should be as small as possible. Since all necessary information to create the TCB is included in the cookie, the server does not need to allocate any client specific resources after the state cookie has been transmitted to the client. This mechanism is a key feature to prevent SCTP INIT-flooding attacks, similar to the TCP SYN-flooding, where an attacker can exhaust the server's resources. On the other hand, this can also be a drawback if the server stores a lot of information in the cookie, for example a long list of supported extensions or many address candidates. A large cookie size may be abused by an attacker for a byte amplification attack or to exhaust the server's uplink capacity. Byte amplification attacks cause the server to send a significantly larger INIT-ACK chunk as a response to an INIT chunk sent by an attacker with a spoofed source address of a victim. In addition to the state cookie, the server can also add any number of optional parameters to the INIT-ACK chunk.

Upon receiving the INIT-ACK chunk, the client returns the received state cookie to the server in a COOKIE-ECHO chunk to authenticate its ownership of the IP address. The server validates the reflected state cookie, creates the TCB, establishes the SCTP association and acknowledges the successful process with a COOKIE-ACK chunk. This COOKIE-ACK chunk opens the association on the client side, the four-way handshake has now successfully finished.

To reduce the timespan between connection initialization and transmitting the first data, the client and the server can bundle DATA chunks with the COOKIE-ECHO chunk and the COOKIE-ACK chunk, respectively, in a single SCTP message, resulting in a timespan of one round-trip for a client between initiating the connection and sending the first data.

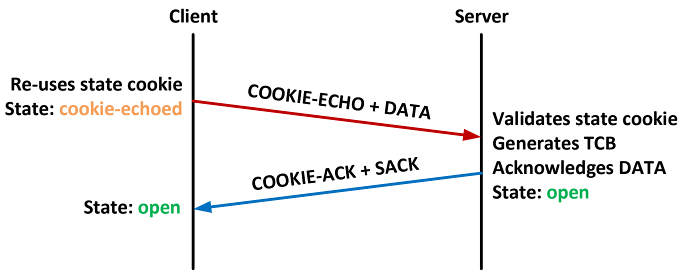

Fig. 3: Zero-RTT connection setup by re-using the state cookie from a previous association

In theory, the client can re-use the state cookie multiple times for future associations with the advantage of saving one round-trip for the connection setup, resulting in a zero-RTT connection setup, which is shown in Figure 3. A disadvantage of re-using the regular state cookie is that all connection specific parameters from the previous association, exchanged by the INIT and INIT-ACK chunk, have to stay the same which affects the *Initiate Tags* and the *Initial Transmission Sequence Numbers* in particular. Re-using the same *Initiate Tag* for more than one association is in conflict with its purpose: providing a key that allows a receiver to verify that the SCTP message belongs to the current association and is not an old or stale message from a previous association and should therefore be unique for every SCTP association. Additionally, the client cannot modify the supported extensions, number of streams or announced IP address candidates. Because of the drawbacks, this method is not implemented in the common network stacks.

III. Alternative Handshake

With our alternative handshake approach, we wanted to create a more lightweight cookie exchange mechanism which reduces the resource consumption on the server side and allows a faster connection setup in less round-trips compared to the regular handshake. Additionally, our alternative handshake mechanism should still be fully backwards compatible to systems without support for the new mechanism.

The alternative handshake, as shown in Figure 4, uses the same SCTP chunks as the regular handshake, only adding new optional parameters. A client using the alternative handshake method initiates an SCTP association by sending a regular INIT chunk to the server. The only difference to a regular handshake is the new ALT-COOKIE parameter which is included in the INIT chunk. By adding this parameter to the INIT chunk, the client announces the support for the alternative handshake and its willingness to use it.

A server, also supporting the alternative handshake extension, who receives the alternative cookie parameter in the INIT chunk, generates a client specific cookie and includes it in an ERROR chunk. The ERROR chunk is sent as a response to the INIT chunk which is silently dropped by the server. This is the first major difference to the regular handshake: instead of generating an INIT-ACK chunk, containing all information the server needs to create the TCB, the server only generates a lightweight cookie to validate the IP address ownership of the client, which is the only purpose of the alternative cookie. The ERROR chunk has a specific error cause ("ALT-COOKIE required") and the client specific cookie in the *Cause-Specific Information* field.

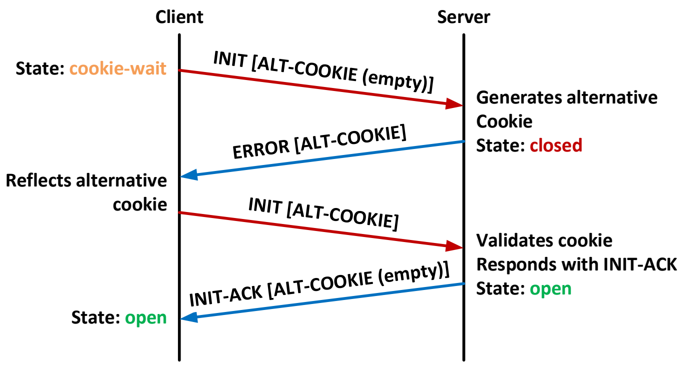

Fig. 4: Alternative four-way handshake

The client responds to the ERROR chunk by transmitting the INIT chunk and, in contrast to the first attempt, including the cookie from the server's ERROR chunk in the INIT chunk's ALT-COOKIE parameter payload field. Upon receiving the INIT chunk with a non-empty ALT-COOKIE parameter, the server validates the alternative cookie and opens the association if successful. In the next step, the server responds with an INIT-ACK chunk which carries an empty ALT-COOKIE parameter and no state cookie to acknowledge the successful usage of the alternative handshake to open the association. This INIT-ACK chunk establishes the association on the client side, and the alternative handshake is finished.

If the server cannot validate the alternative cookie, for whatever reason, it should generate a new alternative cookie and respond with an ERROR chunk, including this new cookie.

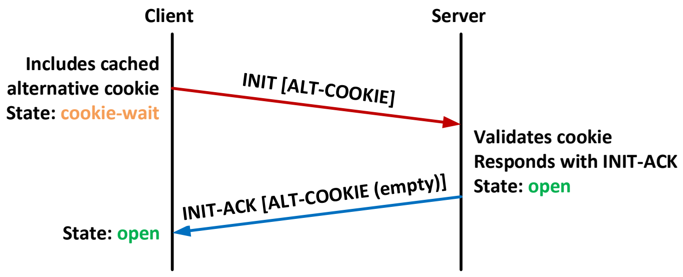

Fig. 5: Alternative handshake using a cached cookie

The client caches the alternative cookie for future associations to the server. To establish new associations, as shown in Figure 5, the client simply includes the previously received alternative cookie in the INIT chunk which instantly opens the association. This method allows an SCTP connection setup in a single round-trip without the disadvantage of re-using association specific parameters like the *Initiate Tags* and the *Initial Transmission Sequence Numbers*.

A. Alternative Cookie Parameter

As already mentioned in the introduction, the INIT and INIT-ACK chunks consist of mandatory and optional parameters. To distinguish the optional parameters, SCTP uses a predefined *Type-Length-Value* (TLV) format to encode the parameter types, variable length and payload.

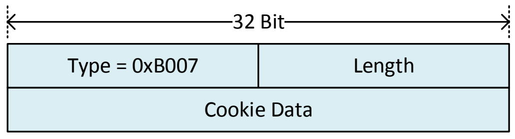

Fig. 6: ALT-COOKIE parameter

The parameter type, represented by a 16-bit field, utilizes the two bits of the highest order to specify the action that must be taken if the processing endpoint does not support the parameter type. The first bit encodes if the endpoint should process any further parameters from the current INIT or INIT-ACK chunk when reading this unknown parameter. If set, an endpoint will stop processing the INIT chunk and report an error. The second bit encodes if an unknown parameter should be reported to the remote endpoint or silently be discarded.

The alternative cookie (ALT-COOKIE) parameter uses this particular TLV structure and is included in the INIT and INIT-ACK chunk for multiple purposes. The ALT-COOKIE parameter type (0xB007) has the highest bit set to one and the second highest bit to zero. This encoding ensures that a server without support for the alternative cookie mechanism will silently discard the ALT-COOKIE parameter and continue processing the SCTP message. The length field contains the size of the parameter in bytes, including the length of the parameter header. An ALT-COOKIE parameter can carry an empty cookie. Therefore, the length value is between 4 and 65535 bytes.

B. Alternative Cookie Calculation

A fundamental requirement in the design process for the alternative cookie was to be more lightweight than the regular state cookie, as already explained in the introduction. This covers the cookie size and its computation time for the server while still offering the same protection level as the regular state cookie. In contrast to the regular state cookie, the only purpose of the alternative cookie is the validation of the client's ownership of the source IP address and, in contrast to the regular state cookie, not to create the TCB on the server side.

The generation and validation of the cookie is in the responsibility of the server. We suggest that an implementation should use an appropriate cryptographic hashing mechanism to ensure the integrity of the cookie. Since the client only reflects the cookie, the server can embed any desired information in the cookie but should keep it as small as possible to mitigate the risk of amplification attacks. Our implementation uses a SIPHASH [11] based keyed-hash message authentication algorithm, which is also used for the TCP Fast Open cookie in the FreeBSD kernel stack, to generate the cookie from the client's IP address. This method binds the alternative cookie to a specific IP address and, therefore, a multihomed client has to use the same path it received the alternative cookie on for future associations. The lifetime of the alternative cookie can be limited by changing the secret key of the cryptographic hash function.

C. Cookieless Handshake

Both handshake methods, the regular and the alternative, aim to prevent amplification attacks and resource exhaustion by malicious peers. But in certain use cases this protection is not necessary, for example if the SCTP communication is transmitted within a protected environment. The usage of SCTP for WebRTC Data Channels represents a typical case for a protected environment due to DTLS encapsulation of the SCTP communication [12]. In this case, the client and the server already perform a DTLS handshake before the first SCTP message is transmitted within the DTLS tunnel. For those scenarios, the alternative handshake method allows an SCTP server to waive the cookie exchange and open the association upon receiving the INIT chunk with an empty ALT-COOKIE parameter.

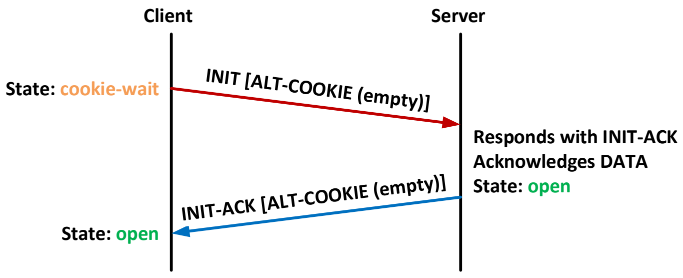

Fig. 7: Cookieless Handshake

Instead of responding with an ERROR chunk, the server acknowledges the successfully established association by sending an INIT-ACK chunk which includes an empty ALT-COOKIE parameter, similar to the last two steps of the alternative handshake with a cookie exchange. This method saves a full round-trip even if both peers have never been in contact before and requires no changes at the client side.

D. Zero-RTT connection setup

In addition to the alternative cookie parameter, our approach introduces an alternative data (ALT-DATA) parameter and an alternative selective acknowledgment (ALT-SACK) parameter. These parameters allow the client to embed application data in the INIT chunk, resulting in a zero RTT connection setup. The ALT-DATA parameter simply acts as a container for regular DATA chunks, see Figure 8. The local endpoint embeds one or more regular DATA chunks in the parameter's value field. The sender has to be aware of not exceeding the path

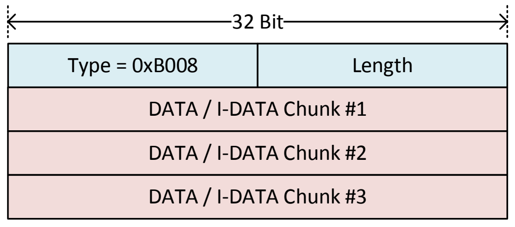

Fig. 8: ALT-DATA parameter

MTU to avoid IP fragmentation when embedding application data.

When the server receives and accepts DATA chunks from the INIT chunk's alternative data parameter, it acknowledges them by an alternative selective acknowledgment parameter (ALT-SACK) which is included in the responding INIT-ACK chunk. This indicates the successful usage of the alternative data parameter to the client.

If the server receives a DATA chunk with an invalid stream number, it drops the DATA chunk according to the procedure specified by RFC4960 and reports it to the client using an ERROR chunk with the *Invalid Stream Identifier* cause. Another case is a server not opening the association and requesting an alternative cookie exchange by sending an ERROR chunk. In this case the application data is also lost and should be retransmitted in the next INIT chunk.

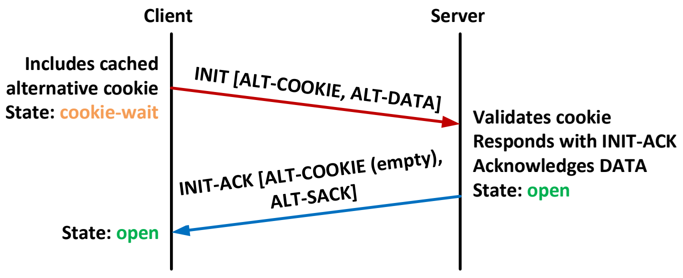

Fig. 9: Alternative Zero-RTT connection setup

A server not supporting data transmission at all via the alternative data parameter will silently discard it and, therefore, not include an ALT-SACK parameter in the responding INIT-ACK chunk. The client retransmits the previously included DATA chunks using the regular way.

E. Fallback Mechanism

A major requirement for the alternative handshake is the seamless backwards compatibility to endpoints not supporting the alternative handshake method, and if the fallback mechanism is used, it should not be unfavorable compared to the regular handshake. As already explained in the previous sections, we use TLV parameters in the INIT chunk to achieve a maximum of compatibility with peers not supporting our new extension. If a server without support for the alternative handshake procedure receives an INIT chunk with an ALT-COOKIE parameter, the server will silently discard the unknown parameter and respond with a regular INIT-ACK chunk not containing the ALT-COOKIE parameter. The client continues with a regular handshake and reflects the received state cookie. When the client connects to the server for the first time, without having a cached cookie, the alternative cookie parameter has no payload and has a length of four bytes.

The behavior of silently ignoring the unsupported parameter is also used for the alternative data parameter. When the initiating peer using the alternative handshake procedure includes application data in the INIT chunk, and the server does not respond with an INIT-ACK chunk with the ALT-COOKIE parameter set and an included ALT-SACK parameter, the application data is lost and has to be retransmitted using regular DATA chunks. In contrast to wasting four bytes by sending an unused ALT-COOKIE parameter, including application data in an INIT chunk should be considered thoroughly because the performance impact may be much higher. In the worst case scenario where the client and the server support the alternative handshake but only the client supports the alternative data parameter, the client will transmit application data three times before it is accepted by the server. The server responds to the first INIT chunk with an ERROR chunk and requests the reflection of the alternative cookie, while dropping the included DATA chunks. The client retransmits the INIT chunk, now including the alternative cookie which opens the association on the server side. The server ignores the alternative data parameter, and the client has to retransmit the application data a third time using the regular DATA chunk.

F. *Multihoming*

A key feature of SCTP is multihoming which allows the usage of multiple IP addresses for a single association, either for automatic failover or load sharing [13]. During the handshake, both endpoints exchange their IP address candidates by including them in the INIT and INIT-ACK chunk, respectively. After the association has been established via the primary path, both peers probe the remote address candidates by sending HEARTBEAT chunks. An endpoint responds to a HEARTBEAT chunk by sending a HEARBEAT-ACK chunk which reflects the *Sender-Specific Heartbeat Info* from the HEARTBEAT chunk.

We have modified this mechanism for the alternative handshake since it leads to problems in certain cases. When a client initiates the association by sending an INIT chunk with the alternative handshake method and the server accepts this INIT chunk, either by successfully validating the cookie or by waiving the cookie exchange, the INIT chunk instantly opens the association on the server side and the server responds with an INIT-ACK chunk. Additionally, the server will probe all the client's IP address candidates by sending HEARTBEAT chunks to the particular IP addresses. If the client receives the server's HEARTBEAT chunk before the INIT-ACK chunk, which may happen if the secondary path is faster than the primary or the HEARTBEAT chunk is processed before the INIT chunk, the client cannot match the HEARTBEAT chunk with an existing association and will respond with an ABORT chunk which terminates the association on the server side. To avoid this behavior, the server will not send HEARTBEAT chunks directly after the association has been established. Instead the server waits until receiving the first additional SCTP message from the client on the new association before sending HEARTBEAT chunks to ensure that the association has successfully been established on the client side.

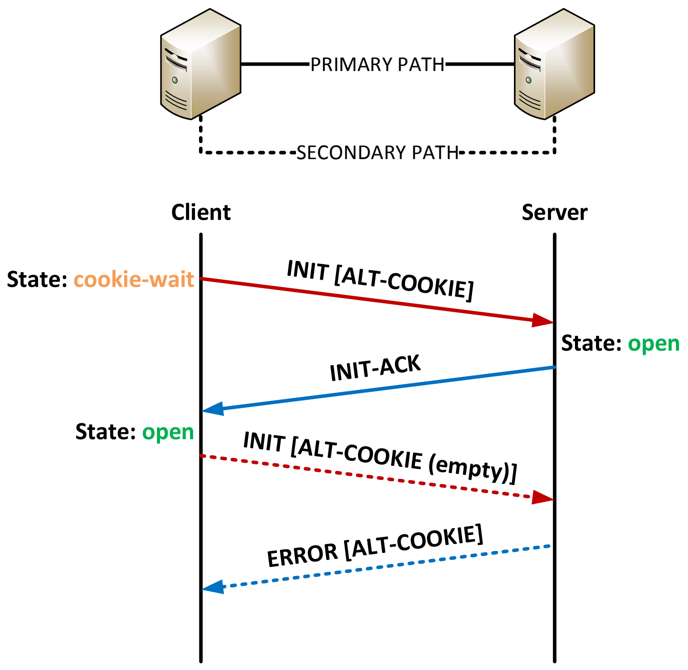

Fig. 10: Alternative cookie exchange on a secondary path

Another change affects the alternative cookie. The alternative cookie only validates the source address of the client from which the INIT chunk has been sent and not the additional address candidates listed in the INIT chunk. Especially mobile devices, like smartphones, often switch between cellular and wifi networks. When establishing a connection to the server, the smartphone uses the wifi network as its primary path and includes its cellular IP address in the INIT chunk as an additional address candidate. When the connection has successfully been established using the primary path, the client has an alternative cookie which validates the ownership of the wifi path but is not valid for a future connection using the cellular path.

To overcome this limitation, the client sends INIT chunks to the server via the additional paths after the association has been established, as shown in Figure 10. The server generates an alternative cookie for the client's source address and responds with an ERROR chunk via the alternative path. The client caches the cookie for future associations on the alternative path.

G. *Initialization Collision*

Initialization collision happens if both peers initiate an SCTP association by sending an INIT chunk simultaneously and use the same address/port combination. The collision handling is an important feature, especially for the WebRTC use-case where both peers take the active part and initiate the SCTP connection simultaneously, as defined in the corresponding IETF draft [14]. The regular handshake has techniques to detect and handle collision cases, but not all of these techniques can be applied to the alternative handshake since the state cookie is lacking.

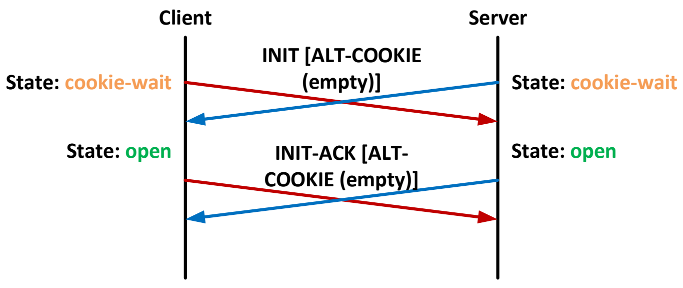

Fig. 11: Initialization collision scenario

When an endpoint receives and accepts an INIT chunk using the alternative handshake procedure from a remote peer while being in a state of waiting for a response for its own INIT chunk from the same address/port combination, as shown in Figure 11, the local endpoint opens the association and responds with an INIT-ACK chunk, following the alternative handshake procedure. But, in contrast to the non-collision case, the local endpoint includes the same parameter as in the previously sent INIT chunk. This especially affects the *initiate tag*. The remote peer receives the INIT-ACK chunk which is carrying the same *initiate tag* parameter as the previously received INIT chunk. This indicates that the local peer has successfully handled the collision case and migrated the colliding connection attempts into a single association.

IV. Extension Compatibility

It is not sufficient to design the alternative handshake mechanism only to match the RFC 4960 since there are additional extensions relying on the traditional handshake mechanism.

A. *User Message Interleaving*

The *User Message Interleaving* (I-DATA) [15] extension is a mandatory part of the WebRTC Data Channel specification [5], solving SCTP's sender side head-of-line blocking issues when sending a large user message which blocks all other streams. During the regular handshake, both peers announce support for the I-DATA extension in the *Supported Extensions Parameter* as defined in RFC 5061 [16]. If the I-DATA extension has been negotiated, which is the case if both peers have announced to support it, both peers must use the I-DATA chunk instead of the DATA chunk. Using the DATA chunk during an association when the I-DATA extension has been negotiated results in an error. The alternative handshake, including the cookieless variant, does not affect the I-DATA negotiation process but if the I-DATA announcing client wants to include application data within the INIT chunk by using the ALT-DATA parameter, the support for this extension has not been negotiated at this point of time. We considered several ways to solve this issue and concluded that the client should proactively use the I-DATA chunk instead of the DATA chunk when including data using the alternative data parameter in the INIT chunk.

When the alternative handshake has successfully finished, the client handles the application data within the alternative data parameter as lost if the I-DATA extension support has not been negotiated and automatically retransmits the application data using a regular DATA chunk.

A server without support for the I-DATA extension will silently discard the I-DATA chunks carried by the alternative data parameter and wait for the client to retransmit the application data using a regular DATA chunk.

B. *Authenticated Chunks*

SCTP's 32-bit verification tags protect the association against a blind attacker but not against a Man-in-the-middle attack where the attacker can easily inject SCTP messages with the correct verification tags. The *Authenticated Chunks* [17] extension solves this issue by allowing the endpoints to authenticate specific peer chunks to verify that the chunks are sent by the remote endpoint and not from a Man-in-the-middle attacker. A sender bundles an authentication chunk with the chunk types which should be authenticated. The authentication chunk contains an HMAC which can be used by the receiver to validate the chunks bundled within the received SCTP message.

The support of this extension and the association specific parameters, like the list of authenticated chunks and the HMAC algorithm, are negotiated during the INIT and INIT-ACK chunk exchange. This procedure is compatible with the alternative handshake using the alternative cookie. However, this extension is not compatible with the usage of the ALT-DATA parameter to include DATA chunks in the INIT chunk because the sending endpoint has no knowledge of the receivers capabilities.

V. Implementation

We have successfully implemented and tested the alternative handshake mechanism for the FreeBSD kernel stack and the SCTP userland implementation (usrsctp) [18]. The userland implementation is supported on all widely used operating systems, including FreeBSD, Linux, macOS and Windows. Additionally, the userland implementation is used for WebRTC Data-Channels in several major browsers, including Google Chrome, Mozilla Firefox, Opera and Apple's Safari.

A. *API extension*

The alternative handshake mechanism is controllable by using the *setsockopt()* and *sysctl()* function calls. The support for the alternative handshake in general is controllable system wide by sysctl variables. System administrators can choose between only allowing the regular handshake (0), the alternative one (2) or both of them (1).

Applications, which want to use the alternative handshake for future associations, use the *setsockopt()* function call to control the functionality per socket. By setting the *SCTP_ALT_HANDSHAKE* socket option, the alternative handshake procedure is activated. The available option values are the same as for the system wide *sysctl* settings. After the association has been established, the application may use the same options for the *getsockopt()* function call to determine if the association has been established using the alternative or the regular handshake.

If the server wants to allow the cookieless alternative handshake, it sets the *SCTP_EMPTY_ALT_COOKIE* socket option on a listening socket. This socket option controls if

```
int sockfd = socket(...);
struct sctp_assoc_value av = { .assoc_id = 0, .assoc_value = 1 };
setsockopt(sockfd, IPPROTO_SCTP, SCTP_ALT_HANDSHAKE, &av, sizeof(av));
setsockopt(sockfd, IPPROTO_SCTP, SCTP_INIT_ALT_DATA, &av, sizeof(av));
char payload[13] = "Hello␣Irene!";
sendto(sockfd, payload, strlen(payload), ...);
```

Listing 1: Sample code for a client using the alternative handshake with zero-RTT connection setup

a server accepts an empty ALT-COOKIE parameter in the INIT chunk as described in the previous section. For existing associations, it allows to query whether the empty cookie method has been used or not on a particular association, e.g. for statistical usage.

If the network socket is configured to support the alternative handshake, the application may bundle application data within the INIT or INIT-ACK chunk using the ALT-DATA parameter. The application can enable this feature by using the *SCTP_INIT_ALT_DATA* option for the *setsockopt()* function call. Since the SCTP network stack needs the application payload before sending the initiate message, application developers cannot use the typical function sequence of *connect()* and *send()*. Calling the *sendto()* function initiates an implicit connection setup and includes the given payload data in the INIT chunk.

Listing 1 shows a simplified SCTP client example using the alternative handshake procedure with application data included in the ALT-DATA parameter of the INIT chunk by using an implicit connection setup.

VI. Measurements and Evaluation

To evaluate the performance of our alternative handshake procedure and its implementation, we created a client-server scenario as shown in Figure 12. Both nodes have the identical hard- and software configuration: PC Engines APU2 boards with mSATA solid state discs and two operating systems installed. We have chosen this hardware configuration by intention to evaluate CPU effects caused by the low performance and energy optimized processor. In addition to FreeBSD HEAD with release type kernel and disabled debugging options (GENERIC-NODEBUG), we used Ubuntu 17.10.

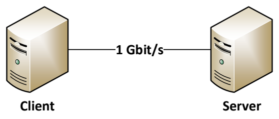

Fig. 12: Testbed for INIT chunk flooding

A. *INIT flooding*

Although SCTP is robust against INIT-flooding attacks, similar to TCP SYN-flooding, however, generating the state cookie consumes a large amount of CPU load for the server side, and the INIT-ACK chunk is significantly larger than the INIT chunk. In a first step, we evaluated the performance of the INIT chunk handling on the server side for different scenarios where a server, running FreeBSD HEAD and Ubuntu 17.10, is flooded with INIT chunks. To send a large amount of SCTP messages containing an INIT chunk while having the greatest possible control over the senders behavior, we developed a benchmark tool using the netmap fast packet I/O framework [19], allowing us to send and receive network packets with a variable rate up to wire speed. To eliminate side effects, we disabled the ethernet flow control on all nodes. The client, running the netmap benchmarking tool, sends SCTP messages containing an INIT chunk with different rates to the server for a fixed timespan of 60 seconds and measures several values during the test run, including the packet-rate, bandwidth and average packet size in both directions. The benchmark tool provides several options to modify the INIT chunks, which includes extensions, address candidates and the number of parameters. We have configured the benchmark tool to flood the server with INIT chunks which announce support for all officially specified extensions and additionally our alternative handshake extension.

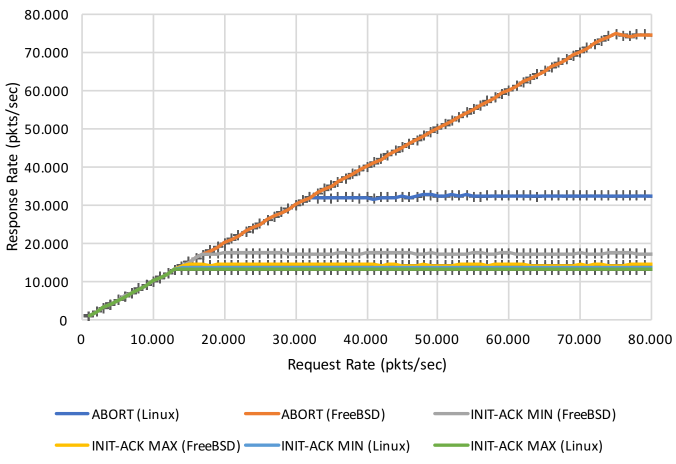

Fig. 13: Comparison of Linux and FreeBSD INIT-ACK rates

In the first test scenario, the client sends small SCTP messages of 112 bytes, containing an INIT chunk, to the server which does not have a listening service on the particular IP/port combination. The server responds with an ABORT chunk which is a cheap operation for the server, its generation consumes only a small amount of resources and its size is only 16 bytes. Therefore, this is our baseline scenario with respect to the responding packet rate and resource consumption on the server side. As shown in Figure 13, we measured an ABORT rate of 74 kpps from the FreeBSD server and an ABORT rate of 32 kpps from the Ubuntu server.

In our second scenario, the server has a listening socket bound to the specific IP/port combination and responds to INIT chunks with an INIT-ACK chunk. As already mentioned in the previous section, the INIT-ACK chunk is significantly larger than the INIT chunk and its generation is more resource consuming. In contrast to Linux, FreeBSD announces all officially specified extensions in the INIT-ACK chunk by default, even if they are not requested by the INIT chunk. To have a more comparable result, we measured the INIT-ACK rate on both machines with two different settings. First with all extensions enabled, labeled as *INIT-ACK MAX* in Figure 13 and additionally with all extensions disabled, labeled as *INIT-ACK MIN*. In contrast to the ABORT rate, the INIT-ACK rates of Ubuntu and FreeBSD are on a similar level. The INIT-ACK rates for both systems range between 13 kpps for an INIT-ACK chunk with all extensions from the Ubuntu machine until 18 kpps INIT-ACK chunks without any extensions from the FreeBSD machine.

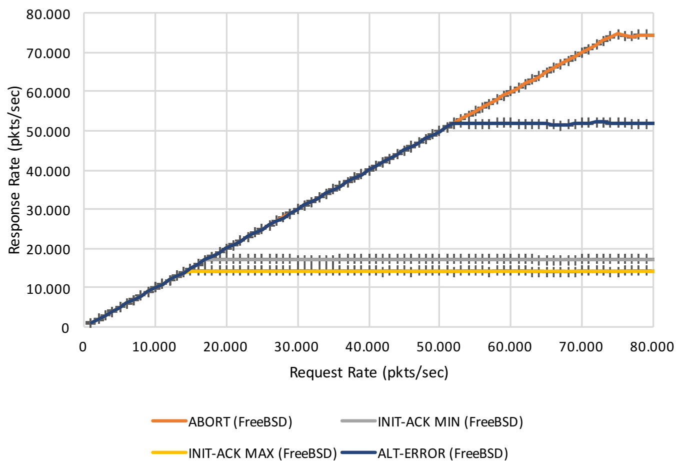

Fig. 14: FreeBSD's response rate to INIT chunks

In a third step we used the alternative handshake method where the server responds with an ERROR chunk, bundled with the alternative cookie. The server is able to respond with about 52k ERROR chunks per second which carry the alternative cookie, shown in Figure 14. In addition, the responses are much smaller compared to the regular INIT-ACK chunk. In our scenario, the regular INIT-ACK chunk, with extensions, has a length of 416 bytes whereas the alternative one has a length of only 40 bytes. While the alternative response is only 4 bytes larger than the initiating request, the regular response is more than ten times larger than the SCTP message containing the INIT chunk. This shows another advantage of the alternative handshake, it successfully prevents byte amplification attacks.

B. *Time to first byte*

A common use case of SCTP, since it is reliable and message oriented, is the transmission of small messages in a request-response manner. Typically the client sends a small request, the server answers with a response and closes the connection afterwards. This traffic pattern is common for signaling services and measurement grids. We developed a client and a server to evaluate the performance improvements of the alternative handshake procedure over the regular handshake.

The client establishes a connection to the server and sends a small request of less than 100 bytes payload. The server also responds with a small message of less than 100 bytes and closes the connection afterwards. We varied the link delay by using the dummynet [20] network emulation tool running on a router between the server and the client, this scenario is shown in Figure 15. The link speed has been set to 2 Mbit/s, and we varied the link delay.

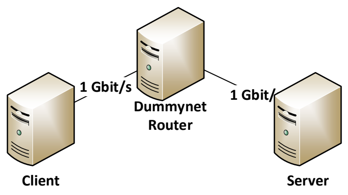

Fig. 15: Testbed for signaling traffic

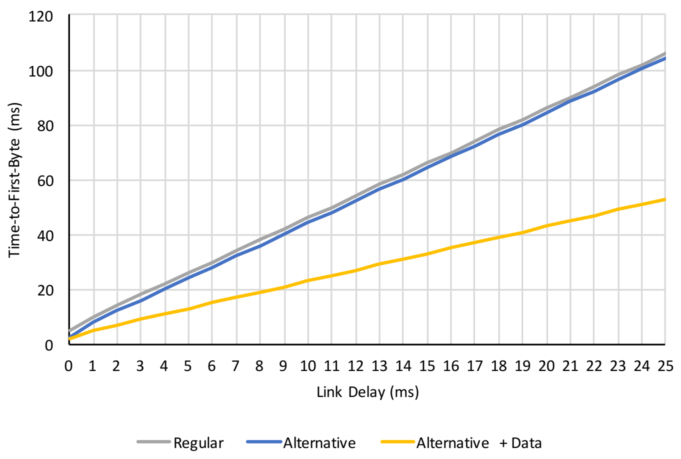

Fig. 16: Timespan between the connection initialization and receiving the first bytes on the client side

Figure 16 shows the results for three different handshake types. We measured the timespan between initiating the association and the arrival of the server's response on the client side. The regular handshake and the alternative handshake show a nearly identical performance, the alternative handshake is slightly faster because of the smaller cookie size. This matches our expectations since both peers are not affected by CPU limitations and both handshake variants take the same amount of round-trips. When the client uses a previously cached cookie, the connection setup time is reduced by one round-trip. Since both DATA chunks, containing the request and the response, fit in a single SCTP message, the timespan until the server's response arrives at the client is reduced by one half when using a previously cached cookie.

C. *Compatibility and Security*

We have tested the backwards compatibility of our new extension with multiple SCTP implementations without support for the extension to ensure its deployability. This includes the implementations of FreeBSD, Linux, Solaris and the userland stack. All of them successfully ignore the alternative parameter in the INIT chunk and continue with the regular handshake by sending an INIT-ACK chunk. Thus, and since introducing new parameters is defined in the official RFC4960, we do not expect any compatibility problems in deployment.

Our measurements and evaluation demonstrates that an attacker is able to exhaust the server's CPU resources by sending a large number of INIT chunks and, since the generated INIT-ACK chunk is mostly larger than the corresponding INIT chunk, may also be used for a byte amplification attack. This insight is not new and has already been treated by RFC5062 [6] in the year 2007, where this kind of attack is characterized as hard to avoid. RFC5062 suggests to use the PAD parameter [21] to artificially enlarge the initiating message and, therefore, prevent this attack pattern.

The QUIC protocol makes use of this method, the initiating QUIC message must at least have a length of 1200 octets. Our implementation allows the server to waive the fallback mechanism and only support the alternative handshake. When configured to do so, a server will always respond with an ERROR chunk including an alternative cookie upon receiving an INIT chunk, even if the client has not announced support for the alternative handshake. This is an effective way to prevent amplification attacks but requires both peers to support the alternative handshake.

Before an application developer enables the new handshake features, their possible drawbacks should be considered carefully. While using the alternative cookie parameter should not have any negative impact, as it has a seamless fallback mechanism and offers the same protection as the regular handshake, the inclusion of application data in the INIT chunk may lead to unwanted behavior regarding security and data integrity. The cookieless handshake should only be enabled in protected environments, like WebRTC Data-Channels.

VII. Conclusion and Outlook

This paper introduces an alternative handshake mechanism for the SCTP protocol which reduces the required round-trips for association establishment and offers a lower resource consumption. Our solution provides the same protection against INIT-flooding attacks as the regular handshake procedure and is fully backwards compatible to peers not supporting the new extension. It can also prevent byte amplification attacks in case the server waives the backwards compatibility by only accepting handshakes using the alternative mode. In certain scenarios, like the usage of SCTP for WebRTC Data-Channels, our new zero RTT connection setup capability gives a significant performance improvement, compared to the regular handshake. We have implemented the new handshake mechanism for the FreeBSD kernel stack and the widely used userland stack and evaluated its impact in several test scenarios with both implementations regarding robustness against INIT-flooding attacks and performance improvements. Our future work will focus on improving the alternative handshake and its implementation for the WebRTC Data-Channel use-case. We are also working on an IETF draft and will publish our implementation.

Acknowledgments

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 644334 (NEAT) and the German Research Foundation (Deutsche Forschungsgemeinschaft). The views expressed are solely those of the authors.

References

[1] Y. Cheng, J. Chu, S. Radhakrishnan, and A. Jain. TCP Fast Open. RFC 7413 (Experimental), December 2014.

[2] Eric Rescorla. The transport layer security (tls) protocol version 1.3. Internet-Draft draft-ietf-tls-tls13-28, IETF Secretariat, July 2017. http://www.ietf.org/internet-drafts/draft-ietf-tls-tls13-28.txt.

[3] Jana Iyengar and Martin Thomson. QUIC: A UDP-Based Multiplexed and Secure Transport. Internet-Draft draft-ietf-quic-transport-10, Internet Engineering Task Force, March 2018. Work in Progress.

[4] R. Stewart (Ed.). Stream Control Transmission Protocol. RFC 4960 (Proposed Standard), September 2007. Updated by RFCs 6096, 6335, 7053.

[5] Randell Jesup, Salvatore Loreto, and Michael Tuexen. Webrtc data channels. Internet-Draft draft-ietf-rtcweb-data-channel-13, IETF Secretariat, January 2015. http://www.ietf.org/internet-drafts/draft-ietf-rtcweb-data-channel-13.txt.

[6] R. Stewart, M. Tuexen, and G. Camarillo. Security Attacks Found Against the Stream Control Transmission Protocol (SCTP) and Current Countermeasures. RFC 5062 (Informational), September 2007.

[7] E. P. Rathgeb, C. Hohendorf, and M. Nordhoff. On the robustness of sctp against dos attacks. In *2008 Third International Conference on Convergence and Hybrid Information Technology*, 2008.

[8] I. Joe and L. Kant. Sctp with an improved cookie mechanism for wireless networks through modeling and simulation. In *2003 IEEE 58th Vehicular Technology Conference. VTC 2003-Fall (IEEE Cat. No.03CH37484)*, 2003.

[9] I. Joe. Sctp with an improved cookie mechanism for mobile ad-hoc networks. In *Global Telecommunications Conference, 2003. GLOBECOM '03. IEEE*, 2003.

[10] E. Rescorla and N. Modadugu. Datagram Transport Layer Security Version 1.2. RFC 6347 (Proposed Standard), January 2012. Updated by RFCs 7507, 7905.

[11] SipHash: a fast short-input PRF. https://131002.net/siphash/. [Online; accessed 24-November-2017].

[12] M. Tuexen, R. Stewart, R. Jesup, and S. Loreto. Datagram Transport Layer Security (DTLS) Encapsulation of SCTP Packets. RFC 8261 (Proposed Standard), November 2017.

[13] Professor Paul D. Amer, Martin Becke, Thomas Dreibholz, Nasif Ekiz, Jana Iyengar, Preethi Natarajan, Randall R. Stewart, and Michael Txen. Load Sharing for the Stream Control Transmission Protocol (SCTP). Internet-Draft draft-tuexen-tsvwg-sctp-multipath-15, Internet Engineering Task Force, January 2018. Work in Progress.

[14] Christer Holmberg, Roman Shpount, Salvatore Loreto, and Gonzalo Camarillo. Session description protocol (sdp) offer/answer procedures for stream control transmission protocol (sctp) over datagram transport layer security (dtls) transport. Internet-Draft draft-ietf-mmusic-sctp-sdp-26, IETF Secretariat, April 2017. http://www.ietf.org/internet-drafts/draft-ietf-mmusic-sctp-sdp-26.txt.

[15] R. Stewart, M. Tuexen, S. Loreto, and R. Seggelmann. Stream Schedulers and User Message Interleaving for the Stream Control Transmission Protocol. RFC 8260 (Proposed Standard), November 2017.

[16] R. Stewart, Q. Xie, M. Tuexen, S. Maruyama, and M. Kozuka. Stream Control Transmission Protocol (SCTP) Dynamic Address Reconfiguration. RFC 5061 (Proposed Standard), September 2007.

[17] M. Tuexen, R. Stewart, P. Lei, and E. Rescorla. Authenticated Chunks for the Stream Control Transmission Protocol (SCTP). RFC 4895 (Proposed Standard), August 2007.

[18] usrsctp - a portable SCTP userland stack. *Available at https://github.com/sctplab/usrsctp*, 2018.

[19] Luigi Rizzo. Netmap: A novel framework for fast packet i/o. In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, USENIX ATC'12, pages 9–9, Berkeley, CA, USA, 2012. USENIX Association.

[20] The dummynet project. http://info.iet.unipi.it/~luigi/dummynet/. [Online; accessed 19-September-2017].

[21] M. Tuexen, R. Stewart, and P. Lei. Padding Chunk and Parameter for the Stream Control Transmission Protocol (SCTP). RFC 4820 (Proposed Standard), March 2007.