Stream:	Internet Engineering Task Force (IETF)
RFC:	8831
Category:	Standards Track
Published:	January 2021
ISSN:	2070-1721
Authors:	R. Jesup	S. Loreto	M. Tüxen
---	---	---	---
	Mozilla	*Ericsson*	*Münster Univ. of Appl. Sciences*

RFC 8831
WebRTC Data Channels

Abstract

The WebRTC framework specifies protocol support for direct, interactive, rich communication using audio, video, and data between two peers' web browsers. This document specifies the non-media data transport aspects of the WebRTC framework. It provides an architectural overview of how the Stream Control Transmission Protocol (SCTP) is used in the WebRTC context as a generic transport service that allows web browsers to exchange generic data from peer to peer.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8831.

Copyright Notice

Table of Contents

1. Introduction

In the WebRTC framework, communication between the parties consists of media (for example, audio and video) and non-media data. Media is sent using the Secure Real-time Transport Protocol (SRTP) and is not specified further here. Non-media data is handled by using the Stream

Control Transmission Protocol (SCTP) [RFC4960] encapsulated in DTLS. DTLS 1.0 is defined in [RFC4347]; the present latest version, DTLS 1.2, is defined in [RFC6347]; and an upcoming version, DTLS 1.3, is defined in [TLS-DTLS13].

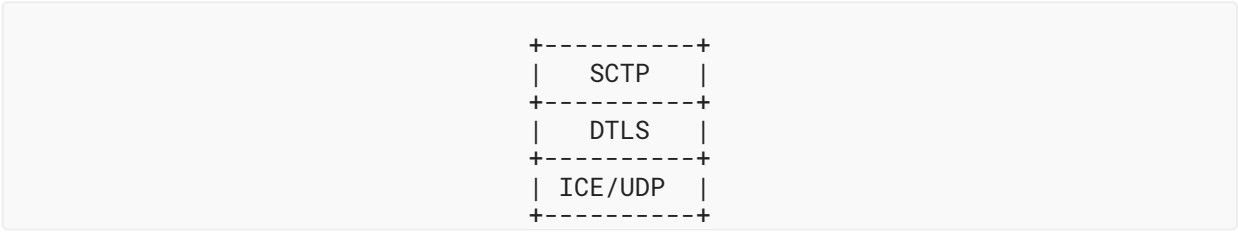

Figure 1: Basic Stack Diagram

The encapsulation of SCTP over DTLS (see [RFC8261]) over ICE/UDP (see [RFC8445]) provides a NAT traversal solution together with confidentiality, source authentication, and integrity-protected transfers. This data transport service operates in parallel to the SRTP media transports, and all of them can eventually share a single UDP port number.

SCTP, as specified in [RFC4960] with the partial reliability extension (PR-SCTP) defined in [RFC3758] and the additional policies defined in [RFC7496], provides multiple streams natively with reliable, and the relevant partially reliable, delivery modes for user messages. Using the reconfiguration extension defined in [RFC6525] allows an increase in the number of streams during the lifetime of an SCTP association and allows individual SCTP streams to be reset. Using [RFC8260] allows the interleave of large messages to avoid monopolization and adds support for prioritizing SCTP streams.

The remainder of this document is organized as follows: Sections 3 and 4 provide use cases and requirements for both unreliable and reliable peer-to-peer data channels; Section 5 discusses SCTP over DTLS over UDP; and Section 6 specifies how SCTP should be used by the WebRTC protocol framework for transporting non-media data between web browsers.

2. Conventions

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Use Cases

This section defines use cases specific to data channels. Please note that this section is informational only.

3.1. Use Cases for Unreliable Data Channels

U-C 1:

A real-time game where position and object state information are sent via one or more unreliable data channels. Note that at any time, there may not be any SRTP media channels or all SRTP media channels may be inactive, and there may also be reliable data channels in use.

U-C 2: Providing non-critical information to a user about the reason for a state update in a video chat or conference, such as mute state.

3.2. Use Cases for Reliable Data Channels

U-C 3: A real-time game where critical state information needs to be transferred, such as control information. Such a game may have no SRTP media channels, or they may be inactive at any given time or may only be added due to in-game actions.

U-C 4: Non-real-time file transfers between people chatting. Note that this may involve a large number of files to transfer sequentially or in parallel, such as when sharing a folder of images or a directory of files.

U-C 5: Real-time text chat during an audio and/or video call with an individual or with multiple people in a conference.

U-C 6: Renegotiation of the configuration of the PeerConnection.

U-C 7: Proxy browsing, where a browser uses data channels of a PeerConnection to send and receive HTTP/HTTPS requests and data, for example, to avoid local Internet filtering or monitoring.

4. Requirements

This section lists the requirements for Peer-to-Peer (P2P) data channels between two browsers. Please note that this section is informational only.

Req. 1: Multiple simultaneous data channels must be supported. Note that there may be zero or more SRTP media streams in parallel with the data channels in the same PeerConnection, and the number and state (active/inactive) of these SRTP media streams may change at any time.

Req. 2: Both reliable and unreliable data channels must be supported.

Req. 3: Data channels of a PeerConnection must be congestion controlled either individually, as a class, or in conjunction with the SRTP media streams of the PeerConnection. This ensures that data channels don't cause congestion problems for these SRTP media streams, and that the WebRTC PeerConnection does not cause excessive problems when run in parallel with TCP connections.

Req. 4: The application should be able to provide guidance as to the relative priority of each data channel relative to each other and relative to the SRTP media streams. This will interact with the congestion control algorithms.

Req. 5: Data channels must be secured, which allows for confidentiality, integrity, and source authentication. See [RFC8826] and [RFC8827] for detailed information.

Req. 6: Data channels must provide message fragmentation support such that IP-layer fragmentation can be avoided no matter how large a message the JavaScript application passes to be sent. It also must ensure that large data channel transfers don't unduly delay traffic on other data channels.

Req. 7: The data channel transport protocol must not encode local IP addresses inside its protocol fields; doing so reveals potentially private information and leads to failure if the address is depended upon.

Req. 8: The data channel transport protocol should support unbounded-length "messages" (i.e., a virtual socket stream) at the application layer for such things as image-file-transfer; implementations might enforce a reasonable message size limit.

Req. 9: The data channel transport protocol should avoid IP fragmentation. It must support Path MTU (PMTU) discovery and must not rely on ICMP or ICMPv6 being generated or being passed back, especially for PMTU discovery.

Req. 10: It must be possible to implement the protocol stack in the user application space.

5. SCTP over DTLS over UDP Considerations

The important features of SCTP in the WebRTC context are the following:

- Usage of TCP-friendly congestion control.
- modifiable congestion control for integration with the SRTP media stream congestion control.
- Support of multiple unidirectional streams, each providing its own notion of ordered message delivery.
- Support of ordered and out-of-order message delivery.
- Support of arbitrarily large user messages by providing fragmentation and reassembly.
- Support of PMTU discovery.
- Support of reliable or partially reliable message transport.

The WebRTC data channel mechanism does not support SCTP multihoming. The SCTP layer will simply act as if it were running on a single-homed host, since that is the abstraction that the DTLS layer (a connection-oriented, unreliable datagram service) exposes.

The encapsulation of SCTP over DTLS defined in [RFC8261] provides confidentiality, source authentication, and integrity-protected transfers. Using DTLS over UDP in combination with Interactive Connectivity Establishment (ICE) [RFC8445] enables middlebox traversal in IPv4- and IPv6-based networks. SCTP as specified in [RFC4960] **MUST** be used in combination with the extension defined in [RFC3758] and provides the following features for transporting non-media data between browsers:

- Support of multiple unidirectional streams.
- Ordered and unordered delivery of user messages.
- Reliable and partially reliable transport of user messages.

Each SCTP user message contains a Payload Protocol Identifier (PPID) that is passed to SCTP by its upper layer on the sending side and provided to its upper layer on the receiving side. The PPID can be used to multiplex/demultiplex multiple upper layers over a single SCTP association. In the WebRTC context, the PPID is used to distinguish between UTF-8 encoded user data, binary-encoded user data, and the Data Channel Establishment Protocol (DCEP) defined in [RFC8832]. Please note that the PPID is not accessible via the JavaScript API.

The encapsulation of SCTP over DTLS, together with the SCTP features listed above, satisfies all the requirements listed in Section 4.

The layering of protocols for WebRTC is shown in Figure 2.

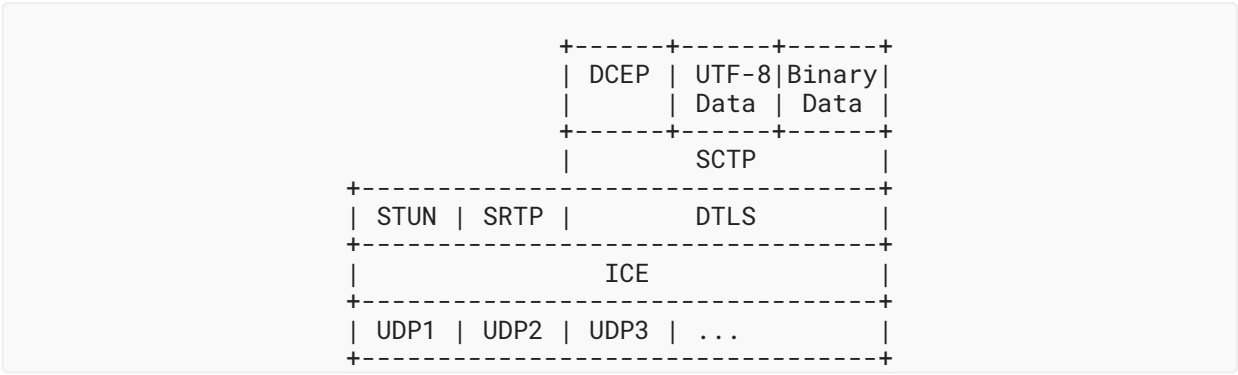

Figure 2: WebRTC Protocol Layers

This stack (especially in contrast to DTLS over SCTP [RFC6083] and in combination with SCTP over UDP [RFC6951]) has been chosen for the following reasons:

- supports the transmission of arbitrarily large user messages;
- shares the DTLS connection with the SRTP media channels of the PeerConnection; and
- provides privacy for the SCTP control information.

Referring to the protocol stack shown in Figure 2:

- the usage of DTLS 1.0 over UDP is specified in [RFC4347];
- the usage of DTLS 1.2 over UDP in specified in [RFC6347];
- the usage of DTLS 1.3 over UDP is specified in an upcoming document [TLS-DTLS13]; and
- the usage of SCTP on top of DTLS is specified in [RFC8261].

Please note that the demultiplexing Session Traversal Utilities for NAT (STUN) [RFC5389] vs. SRTP vs. DTLS is done as described in Section 5.1.2 of [RFC5764], and SCTP is the only payload of DTLS.

Since DTLS is typically implemented in user application space, the SCTP stack also needs to be a user application space stack.

The ICE/UDP layer can handle IP address changes during a session without needing interaction with the DTLS and SCTP layers. However, SCTP **SHOULD** be notified when an address change has happened. In this case, SCTP **SHOULD** retest the Path MTU and reset the congestion state to the initial state. In the case of window-based congestion control like the one specified in [RFC4960], this means setting the congestion window and slow-start threshold to its initial values.

Incoming ICMP or ICMPv6 messages can't be processed by the SCTP layer, since there is no way to identify the corresponding association. Therefore, SCTP **MUST** support performing Path MTU discovery without relying on ICMP or ICMPv6 as specified in [RFC4821] by using probing messages specified in [RFC4820]. The initial Path MTU at the IP layer **SHOULD NOT** exceed 1200 bytes for IPv4 and 1280 bytes for IPv6.

In general, the lower-layer interface of an SCTP implementation should be adapted to address the differences between IPv4 and IPv6 (being connectionless) or DTLS (being connection oriented).

When the protocol stack shown in Figure 2 is used, DTLS protects the complete SCTP packet, so it provides confidentiality, integrity, and source authentication of the complete SCTP packet.

SCTP provides congestion control on a per-association basis. This means that all SCTP streams within a single SCTP association share the same congestion window. Traffic not being sent over SCTP is not covered by SCTP congestion control. Using a congestion control different from the standard one might improve the impact on the parallel SRTP media streams.

SCTP uses the same port number concept as TCP and UDP. Therefore, an SCTP association uses two port numbers, one at each SCTP endpoint.

6. The Usage of SCTP for Data Channels

6.1. SCTP Protocol Considerations

The DTLS encapsulation of SCTP packets as described in [RFC8261] **MUST** be used.

This SCTP stack and its upper layer **MUST** support the usage of multiple SCTP streams. A user message can be sent ordered or unordered and with partial or full reliability.

The following SCTP protocol extensions are required:

- The stream reconfiguration extension defined in [RFC6525] **MUST** be supported. It is used for closing channels.
- The dynamic address reconfiguration extension defined in [RFC5061] **MUST** be used to signal the support of the stream reset extension defined in [RFC6525]. Other features of [RFC5061] are **OPTIONAL**.
- The partial reliability extension defined in [RFC3758] **MUST** be supported. In addition to the timed reliability PR-SCTP policy defined in [RFC3758], the limited retransmission policy defined in [RFC7496] **MUST** be supported. Limiting the number of retransmissions to zero, combined with unordered delivery, provides a UDP-like service where each user message is sent exactly once and delivered in the order received.

The support for message interleaving as defined in [RFC8260] **SHOULD** be used.

6.2. SCTP Association Management

In the WebRTC context, the SCTP association will be set up when the two endpoints of the WebRTC PeerConnection agree on opening it, as negotiated by the JavaScript Session Establishment Protocol (JSEP), which is typically an exchange of the Session Description Protocol (SDP) [RFC8829]. It will use the DTLS connection selected via ICE, and typically this will be shared via BUNDLE or equivalent with DTLS connections used to key the SRTP media streams.

The number of streams negotiated during SCTP association setup **SHOULD** be 65535, which is the maximum number of streams that can be negotiated during the association setup.

SCTP supports two ways of terminating an SCTP association. The first method is a graceful one, where a procedure that ensures no messages are lost during the shutdown of the association is used. The second method is a non-graceful one, where one side can just abort the association.

Each SCTP endpoint continuously supervises the reachability of its peer by monitoring the number of retransmissions of user messages and test messages. In case of excessive retransmissions, the association is terminated in a non-graceful way.

If an SCTP association is closed in a graceful way, all of its data channels are closed. In case of a non-graceful teardown, all data channels are also closed, but an error indication **SHOULD** be provided if possible.

6.3. SCTP Streams

SCTP defines a stream as a unidirectional logical channel existing within an SCTP association to another SCTP endpoint. The streams are used to provide the notion of in-sequence delivery and for multiplexing. Each user message is sent on a particular stream, either ordered or unordered. Ordering is preserved only for ordered messages sent on the same stream.

6.4. Data Channel Definition

Data channels are defined such that their accompanying application-level API can closely mirror the API for WebSockets, which implies bidirectional streams of data and a textual field called 'label' used to identify the meaning of the data channel.

The realization of a data channel is a pair of one incoming stream and one outgoing SCTP stream having the same SCTP stream identifier. How these SCTP stream identifiers are selected is protocol and implementation dependent. This allows a bidirectional communication.

Additionally, each data channel has the following properties in each direction:

- reliable or unreliable message transmission: In case of unreliable transmissions, the same level of unreliability is used. Note that, in SCTP, this is a property of an SCTP user message and not of an SCTP stream.

- in-order or out-of-order message delivery for message sent: Note that, in SCTP, this is a property of an SCTP user message and not of an SCTP stream.
- a priority, which is a 2-byte unsigned integer: These priorities **MUST** be interpreted as weighted-fair-queuing scheduling priorities per the definition of the corresponding stream scheduler supporting interleaving in [RFC8260]. For use in WebRTC, the values used **SHOULD** be one of 128 ("below normal"), 256 ("normal"), 512 ("high"), or 1024 ("extra high").
- an optional label.
- an optional protocol.

Note that for a data channel being negotiated with the protocol specified in [RFC8832], all of the above properties are the same in both directions.

6.5. Opening a Data Channel

Data channels can be opened by using negotiation within the SCTP association (called in-band negotiation) or out-of-band negotiation. Out-of-band negotiation is defined as any method that results in an agreement as to the parameters of a channel and the creation thereof. The details are out of scope of this document. Applications using data channels need to use the negotiation methods consistently on both endpoints.

A simple protocol for in-band negotiation is specified in [RFC8832].

When one side wants to open a channel using out-of-band negotiation, it picks a stream. Unless otherwise defined or negotiated, the streams are picked based on the DTLS role (the client picks even stream identifiers, and the server picks odd stream identifiers). However, the application is responsible for avoiding collisions with existing streams. If it attempts to reuse a stream that is part of an existing data channel, the addition **MUST** fail. In addition to choosing a stream, the application **SHOULD** also determine the options to be used for sending messages. The application **MUST** ensure in an application-specific manner that the application at the peer will also know the selected stream to be used, as well as the options for sending data from that side.

6.6. Transferring User Data on a Data Channel

All data sent on a data channel in both directions **MUST** be sent over the underlying stream using the reliability defined when the data channel was opened, unless the options are changed or per-message options are specified by a higher level.

The message orientation of SCTP is used to preserve the message boundaries of user messages. Therefore, senders **MUST NOT** put more than one application message into an SCTP user message. Unless the deprecated PPID-based fragmentation and reassembly is used, the sender **MUST** include exactly one application message in each SCTP user message.

The SCTP Payload Protocol Identifiers (PPIDs) are used to signal the interpretation of the "payload data". The following PPIDs **MUST** be used (see Section 8):

WebRTC String: to identify a non-empty JavaScript string encoded in UTF-8.

WebRTC String Empty: to identify an empty JavaScript string encoded in UTF-8.

WebRTC Binary: to identify non-empty JavaScript binary data (ArrayBuffer, ArrayBufferView, or Blob).

WebRTC Binary Empty: to identify empty JavaScript binary data (ArrayBuffer, ArrayBufferView, or Blob).

SCTP does not support the sending of empty user messages. Therefore, if an empty message has to be sent, the appropriate PPID (WebRTC String Empty or WebRTC Binary Empty) is used, and the SCTP user message of one zero byte is sent. When receiving an SCTP user message with one of these PPIDs, the receiver **MUST** ignore the SCTP user message and process it as an empty message.

The usage of the PPIDs "WebRTC String Partial" and "WebRTC Binary Partial" is deprecated. They were used for a PPID-based fragmentation and reassembly of user messages belonging to reliable and ordered data channels.

If a message with an unsupported PPID is received or some error condition related to the received message is detected by the receiver (for example, illegal ordering), the receiver **SHOULD** close the corresponding data channel. This implies in particular that extensions using additional PPIDs can't be used without prior negotiation.

The SCTP base protocol specified in [RFC4960] does not support the interleaving of user messages. Therefore, sending a large user message can monopolize the SCTP association. To overcome this limitation, [RFC8260] defines an extension to support message interleaving, which **SHOULD** be used. As long as message interleaving is not supported, the sender **SHOULD** limit the maximum message size to 16 KB to avoid monopolization.

It is recommended that the message size be kept within certain size bounds, as applications will not be able to support arbitrarily large single messages. This limit has to be negotiated, for example, by using [RFC8841].

The sender **SHOULD** disable the Nagle algorithm (see [RFC1122]) to minimize the latency.

6.7. Closing a Data Channel

Closing of a data channel **MUST** be signaled by resetting the corresponding outgoing streams [RFC6525]. This means that if one side decides to close the data channel, it resets the corresponding outgoing stream. When the peer sees that an incoming stream was reset, it also resets its corresponding outgoing stream. Once this is completed, the data channel is closed. Resetting a stream sets the Stream Sequence Numbers (SSNs) of the stream back to 'zero' with a corresponding notification to the application layer that the reset has been performed. Streams are available for reuse after a reset has been performed.

[RFC6525] also guarantees that all the messages are delivered (or abandoned) before the stream is reset.

7. Security Considerations

This document does not add any additional considerations to the ones given in [RFC8826] and [RFC8827].

It should be noted that a receiver must be prepared for a sender that tries to send arbitrarily large messages.

8. IANA Considerations

This document uses six already registered SCTP Payload Protocol Identifiers (PPIDs): "DOMString Last", "Binary Data Partial", "Binary Data Last", "DOMString Partial", "WebRTC String Empty", and "WebRTC Binary Empty". [RFC4960] creates the "SCTP Payload Protocol Identifiers" registry from which these identifiers were assigned. IANA has updated the reference of these six assignments to point to this document and changed the names of the first four PPIDs. The corresponding dates remain unchanged.

The six assignments have been updated to read:

Value	SCTP PPID	Reference	Date
WebRTC String	51	RFC 8831	2013-09-20
WebRTC Binary Partial (deprecated)	52	RFC 8831	2013-09-20
WebRTC Binary	53	RFC 8831	2013-09-20
WebRTC String Partial (deprecated)	54	RFC 8831	2013-09-20
WebRTC String Empty	56	RFC 8831	2014-08-22
WebRTC Binary Empty	57	RFC 8831	2014-08-22

Table 1

9. References

9.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3758] Stewart, R., Ramalho, M., Xie, Q., Tuexen, M., and P. Conrad, "Stream Control Transmission Protocol (SCTP) Partial Reliability Extension", RFC 3758, DOI 10.17487/RFC3758, May 2004, <https://www.rfc-editor.org/info/rfc3758>.

[RFC4820] Tuexen, M., Stewart, R., and P. Lei, "Padding Chunk and Parameter for the Stream Control Transmission Protocol (SCTP)", RFC 4820, DOI 10.17487/RFC4820, March 2007, <https://www.rfc-editor.org/info/rfc4820>.

[RFC4821] Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery", RFC 4821, DOI 10.17487/RFC4821, March 2007, <https://www.rfc-editor.org/info/rfc4821>.

[RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <https://www.rfc-editor.org/info/rfc4960>.

[RFC5061] Stewart, R., Xie, Q., Tuexen, M., Maruyama, S., and M. Kozuka, "Stream Control Transmission Protocol (SCTP) Dynamic Address Reconfiguration", RFC 5061, DOI 10.17487/RFC5061, September 2007, <https://www.rfc-editor.org/info/rfc5061>.

[RFC6525] Stewart, R., Tuexen, M., and P. Lei, "Stream Control Transmission Protocol (SCTP) Stream Reconfiguration", RFC 6525, DOI 10.17487/RFC6525, February 2012, <https://www.rfc-editor.org/info/rfc6525>.

[RFC7496] Tuexen, M., Seggelmann, R., Stewart, R., and S. Loreto, "Additional Policies for the Partially Reliable Stream Control Transmission Protocol Extension", RFC 7496, DOI 10.17487/RFC7496, April 2015, <https://www.rfc-editor.org/info/rfc7496>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8260] Stewart, R., Tuexen, M., Loreto, S., and R. Seggelmann, "Stream Schedulers and User Message Interleaving for the Stream Control Transmission Protocol", RFC 8260, DOI 10.17487/RFC8260, November 2017, <https://www.rfc-editor.org/info/rfc8260>.

[RFC8261] Tuexen, M., Stewart, R., Jesup, R., and S. Loreto, "Datagram Transport Layer Security (DTLS) Encapsulation of SCTP Packets", RFC 8261, DOI 10.17487/RFC8261, November 2017, <https://www.rfc-editor.org/info/rfc8261>.

[RFC8445] Keranen, A., Holmberg, C., and J. Rosenberg, "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal", RFC 8445, DOI 10.17487/RFC8445, July 2018, <https://www.rfc-editor.org/info/rfc8445>.

[RFC8826] Rescorla, E., "Security Considerations for WebRTC", RFC 8826, DOI 10.17487/RFC8826, January 2021, <https://www.rfc-editor.org/info/rfc8826>.

[RFC8827] Rescorla, E., "WebRTC Security Architecture", RFC 8827, DOI 10.17487/RFC8827, January 2021, <https://www.rfc-editor.org/info/rfc8827>.

[RFC8829] Uberti, J., Jennings, C., and E. Rescorla, Ed., "JavaScript Session Establishment Protocol (JSEP)", RFC 8829, DOI 10.17487/RFC8829, January 2021, <https://www.rfc-editor.org/info/rfc8829>.

[RFC8832] Jesup, R., Loreto, S., and M. Tüxen, "WebRTC Data Channel Establishment Protocol", RFC 8832, DOI 10.17487/RFC8832, January 2021, <https://www.rfc-editor.org/info/rfc8832>.

[RFC8841] Holmberg, C., Shpount, R., Loreto, S., and G. Camarillo, "Session Description Protocol (SDP) Offer/Answer Procedures for Stream Control Transmission Protocol (SCTP) over Datagram Transport Layer Security (DTLS) Transport", RFC 8841, DOI 10.17487/RFC8841, January 2021, <https://www.rfc-editor.org/info/rfc8841>.

9.2. Informative References

[RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <https://www.rfc-editor.org/info/rfc1122>.

[RFC4347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security", RFC 4347, DOI 10.17487/RFC4347, April 2006, <https://www.rfc-editor.org/info/rfc4347>.

[RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, DOI 10.17487/RFC5389, October 2008, <https://www.rfc-editor.org/info/rfc5389>.

[RFC5764] McGrew, D. and E. Rescorla, "Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP)", RFC 5764, DOI 10.17487/RFC5764, May 2010, <https://www.rfc-editor.org/info/rfc5764>.

[RFC6083] Tuexen, M., Seggelmann, R., and E. Rescorla, "Datagram Transport Layer Security (DTLS) for Stream Control Transmission Protocol (SCTP)", RFC 6083, DOI 10.17487/RFC6083, January 2011, <https://www.rfc-editor.org/info/rfc6083>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <https://www.rfc-editor.org/info/rfc6347>.

[RFC6951] Tuexen, M. and R. Stewart, "UDP Encapsulation of Stream Control Transmission Protocol (SCTP) Packets for End-Host to End-Host Communication", RFC 6951, DOI 10.17487/RFC6951, May 2013, <https://www.rfc-editor.org/info/rfc6951>.

[TLS-DTLS13] Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", Work in Progress, Internet-Draft, draft-ietf-tls-dtls13-39, 2 November 2020, <https://tools.ietf.org/html/draft-ietf-tls-dtls13-39>.

Acknowledgements

Many thanks for comments, ideas, and text from Harald Alvestrand, Richard Barnes, Adam Bergkvist, Alissa Cooper, Benoit Claise, Spencer Dawkins, Gunnar Hellström, Christer Holmberg, Cullen Jennings, Paul Kyzivat, Eric Rescorla, Adam Roach, Irene Rüngeler, Randall Stewart, Martin Stiemerling, Justin Uberti, and Magnus Westerlund.

Authors' Addresses

Randell Jesup
Mozilla
United States of America
Email: randell-ietf@jesup.org

Salvatore Loreto
Ericsson
Hirsalantie 11
FI-02420 Jorvas
Finland
Email: salvatore.loreto@ericsson.com

Michael Tüxen
Münster University of Applied Sciences
Stegerwaldstrasse 39
48565 Steinfurt
Germany
Email: tuexen@fh-muenster.de