Audio/Video Transport Core Maintenance
Internet-Draft
Intended status: Informational
Expires: 3 September 2026

P. Hancke
Meta Platforms Inc.
J. Uberti
OpenAI
J. Oreland
Google

2 March 2026

STUN Protocol for Embedding DTLS (SPED)
draft-hancke-webrtc-sped-00

Abstract

WebRTC setup normally serializes ICE and DTLS, adding at least one extra round trip before secure media can flow. This document defines the STUN Protocol for Embedding DTLS (SPED), which carries DTLS handshake data and acknowledgements inside STUN Binding Requests and Responses. SPED allows ICE and DTLS to proceed in parallel, improves setup behavior under loss, and remains backward compatible with existing ICE processing.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://fippo.github.io/warp-snap-sped/draft-hancke-webrtc-sped-00.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-hancke-webrtc-sped/.

Discussion of this document takes place on the AVTCORE Working Group mailing list (mailto:avt@ietf.org), which is archived at https://datatracker.ietf.org/wg/avtcore/. Subscribe at https://www.ietf.org/mailman/listinfo/avt/.

Source for this draft and an issue tracker can be found at https://github.com/fippo/warp-snap-sped.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

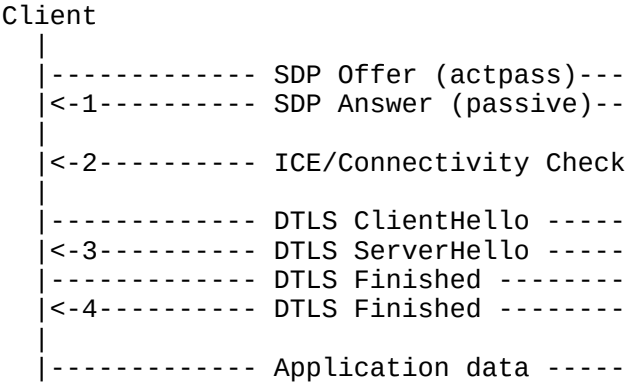
Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Copyright Notice

Table of Contents

1. Introduction

The current WebRTC connection setup, as outlined in [RFC8829], incurs a minimum of 4 RTTs with DTLS 1.2, or 3 RTTs with DTLS 1.3, before media can be sent. The serialization of ICE and DTLS is a large contributor to that as illustrated below for DTLS 1.2:



In addition, deployment experience has shown connection setup reliability issues in scenarios with packet loss, caused by the exponential backoff timer typically used in DTLS implementations.

The protocol defined in this specification, SPED, aims to resolve these concerns by embedding the DTLS handshake into STUN, eliminating the delay caused by the serialization of the protocols and improving reliability by sending fewer packets as well as simplifying retransmissions. In fact, when DTLS 1.3 is used, the protocol can reduce the setup latency to as little as a single round-trip, comparable to the latency of the largely deprecated SDES key exchange mechanism [RFC4568].

The protocol is backward compatible, supports both DTLS 1.2 [RFC6347] and DTLS 1.3 [RFC9147], and can accommodate all DTLS cipher suites, including post-quantum cryptography (PQC) suites that can increase the number of packets sent during DTLS handshaking.

2. Conventions and Definitions

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. Design

3.1. Background

3.1.1. ICE Overview

The ICE protocol [RFC8445] is complex, but the core steps taken by each ICE agent (client) can be summarized as follows:

1. Enumerate local ICE candidates and send them, out-of-band, to the peer.

2. Combine local ICE candidates with the received remote ICE candidates to form ICE candidate pairs.

3. Evaluate the usability of these ICE candidate pairs by sending STUN Binding Requests. This typically happens in parallel, i.e. an ICE agent may have several binding requests in flight when there are multiple candidate pairs.

4. When a STUN Binding Request is received, reply with a STUN Binding Response.

5. When a STUN Binding Response is received, in response to a request, mark the associated ICE candidate pair as valid.

6. If the ICE agent is in the controlling role, select the "best" ICE candidate pair for subsequent sending of data or media, and indicate the selected candidate pair to the remote ICE agent by sending a new STUN Binding Request with the USE-CANDIDATE flag set.

Some endpoints, typically servers, implement a simpler form of ICE known as ICE Lite [RFC8445]. When this form of ICE is used, the ICE Lite endpoint omits steps 3 and 5, and Binding Requests only flow in one direction, from the full to the lite endpoint.

3.1.2. DTLS Overview

In WebRTC, DTLS handshaking normally starts once ICE has identified a valid candidate pair, using "client" and "server" roles determined through the a=setup attribute in WebRTC SDP signaling [RFC5763]. SPED changes when these DTLS packets can be sent, but not the DTLS handshake contents themselves.

We define the term "DTLS packet" to mean the unit of DTLS data typically carried in a single UDP packet.

DTLS handshake messages are also organized into "DTLS flights", as detailed in Section 5.7 of [RFC9147]. A DTLS flight consists of a set of handshake messages that are sent together by a DTLS endpoint, and those messages are carried in one or more DTLS packets.

Ideally, a flight, even if it contains multiple messages, can fit into a single DTLS packet. However, if a message is large, for example a large certificate, it can be fragmented across multiple DTLS packets.

The DTLS flights used during WebRTC session setup are described below. Note that because ICE has already demonstrated remote consent, DTLS' HelloVerifyRequest is not needed to prevent DoS attacks.

Once the DTLS handshake has completed, SRTP key extraction occurs and is used to key the sending of media [RFC5764]. Media cannot be properly decrypted until all handshake messages have been received.

3.1.2.1. DTLS 1.2 Handshake

The DTLS 1.2 handshake, as specified in Section 4.2 of [RFC6347], is organized into the following DTLS flights:

1. The DTLS client sends the ClientHello message.

2. The DTLS server responds with the ServerHello, Certificate, ServerKeyExchange, CertificateRequest, and ServerHelloDone messages, packed as noted above.

3. The DTLS client sends the Certificate, ClientKeyExchange, CertificateVerify, ChangeCipherSpec, and Finished messages.

4. The DTLS server sends the ChangeCipherSpec and Finished messages.

Note that DTLS 1.2 does not provide a separate acknowledgement for the server's final flight, so determination of handshake completion must be done implicitly, e.g., via receipt of application data or expiration of retransmission timers.

3.1.2.2. DTLS 1.3 Handshake

The DTLS 1.3 handshake, as specified in Section 5 of [RFC9147], is organized into the following DTLS flights:

1. The DTLS client sends the ClientHello message.

2. The DTLS server sends the ServerHello, EncryptedExtensions, CertificateRequest, Certificate, CertificateVerify, and Finished messages.

3. The DTLS client sends the Certificate, CertificateVerify, and Finished messages.

Note that in DTLS 1.3, the DTLS server sends a DTLS acknowledgement packet upon receiving the Finished message, but the client does not need to wait for this message to begin sending encrypted data.

3.2. Goals

The desired properties of this solution are:

* It makes WebRTC setup faster by 1 RTT, by allowing ICE and DTLS to proceed in parallel.

* It makes WebRTC session setup less susceptible to packet loss.

* It is strictly an optimization.

* It reduces the number of packets exchanged during session setup, but the size of the STUN Binding Request or Response increases.

* In the event of an incompatibility, each client proceeds with ICE and DTLS as usual.

* It works with all versions of DTLS >= 1.2, and all DTLS cipher suites.

* It is fully backward compatible with existing ICE processing, including interactions with ICE Lite endpoints, as well as endpoints that demultiplex multiple ICE sessions on the same port.

3.3. SPED Protocol

3.3.1. Summary

The overall mechanism can be summarized as follows:

1. DTLS is started at the same time as ICE.

2. If there is no valid ICE candidate pair, DTLS handshake packets are sent by encapsulating them in a new STUN attribute in the next STUN Binding Request or STUN Binding Response.

3. Once a valid ICE candidate pair exists, the client can continue to send DTLS packets either in embedded form, or as usual over the specified pair.

In addition, to improve the reliability of the DTLS handshake, an explicit acknowledgement mechanism is built into SPED. Encapsulated DTLS handshake packets are acknowledged by sending their CRC-32 in a new STUN attribute in the next STUN Binding Request or STUN Binding Response.

3.3.2. New STUN Attributes

This STUN extension defines the following new IETF-assigned attributes in the comprehension-optional range:

* TBD1: DTLS-IN-STUN-DATA

* TBD2: DTLS-IN-STUN-ACK

These attributes have lengths that are not always multiples of 4. By the rules of STUN, any attribute whose length is not a multiple of 4 bytes MUST be immediately followed by 1 to 3 padding bytes to ensure the next attribute, if any, starts on a 4-byte boundary; see [RFC5389].

3.3.2.1. DTLS-IN-STUN-DATA

* This attribute contains one DTLS handshake packet, or is empty to indicate SPED support when no DTLS packet is being embedded.

* While SPED is active, this attribute MUST be present in every STUN Binding Request or Response sent by a SPED-capable agent.

* The value portion of this attribute is variable length and consists of one DTLS handshake packet from a DTLS flight, as described in Section 5.1 of [RFC9147] or Section 4.2 of [RFC6347].

* As noted, if the attribute length is not a multiple of 4, padding must be added.

* If the value portion of this attribute is empty, it indicates SPED support and that no DTLS packet is being embedded in that STUN message. An empty value MUST NOT be injected into the DTLS layer.

* If the value portion of this attribute is non-empty but the first byte is not DTLS, i.e. between 20 and 63 inclusive as described in Section 3 of [RFC9443], the attribute SHOULD be silently discarded.

3.3.2.2. DTLS-IN-STUN-ACK

* This attribute contains acknowledgements of received DTLS-IN-STUN-DATA packets in the order they were received.

* The attribute can be present in either a STUN Binding Request or Response.

* The attribute is variable length and contains a list of uint32 entries, where each entry is the computed CRC-32 of a received DTLS-IN-STUN-DATA attribute value, i.e. a DTLS handshake packet, ignoring padding.

* Implementations SHOULD cap the number of uint32 entries included in this attribute. A cap of 4 entries is RECOMMENDED, which bounds the attribute size while still covering all known handshake cases.

* The attribute can be empty, i.e. the length of the list of uint32 values can be 0.

3.3.3. MTU Considerations

When embedding DTLS in STUN, the DTLS MTU MUST take into account the STUN packet overhead, which is noted in the table below:

Attribute	Size	Defined in
STUN header	20	[RFC5389]
ICE-CONTROLLED / ICE-CONTROLLING	12	Section 19.1 of [RFC5245]
PRIORITY	8	Section 19.1 of [RFC5245]
USE-CANDIDATE	4	Section 19.1 of [RFC5245]; not on first packet but on subsequent packets
MESSAGE-INTEGRITY	24	Section 15.4 of [RFC5389]
MESSAGE-INTEGRITY-SHA256	36	Section 14.6 of [RFC8489]; only applicable when ice2 is used Section 10 of [RFC8445]
FINGERPRINT	8	Section 15.5 of [RFC5389]
DTLS-IN-STUN-DATA	4	This specification. Overhead for the attribute header
DTLS-IN-STUN-ACK	4-20	This specification. 4 byte attribute header plus 0 to 16 bytes for 0 to 4 CRC-32 values under the RECOMMENDED cap
USERNAME	16+	Section 7.1.2.3 of [RFC5245]. Variable, typically 4 byte header plus 9 bytes for two four-byte username fragments and the colon plus 3 bytes padding. The actual size is known before the DTLS exchange starts, either from the SDP exchange or a peer-reflexive candidate
TURN XOR-PEER-ADDRESS	24	[RFC8656]. Assuming 16 byte IPv6; only applicable when TURN is used

Table 1

Accordingly, the typical 1200 byte DTLS MTU, based on the recommendation in [RFC8831], MUST be reduced by the size of the expected overhead. Applications that use custom STUN attributes, i.e. not in the table above, MUST reduce the DTLS MTU further.

3.3.4. Backwards Compatibility

SPED is fully backwards compatible with existing ICE agents. If the peer ICE agent does not support SPED, this can be detected via the lack of the mandatory DTLS-IN-STUN-DATA attribute in its first authenticated ICE check or response, and upon recognizing this fact the local ICE agent can easily fall back to standard unencapsulated DTLS.

Given this straightforward in-band negotiation, this specification does not currently define an offer/answer negotiation mechanism or any ICE options. Note that even if an ICE option were used, the offerer would still need to be prepared to handle ICE checks, with or without SPED, that arrive before the signaling answer.

4. Mechanism

The specifics of the SPED algorithm are detailed below.

4.1. Setup

When using SPED, an ICE agent keeps two lists:

1. A list, L1, of pending DTLS handshake packets.

   These packets are created by the DTLS layer. Elements in the list are removed when ACKed by the peer. The list is cleared when the DTLS layer creates a new flight or the DTLS handshake completes.

2. A list, L2, of pending acknowledgements, as defined above.

   Entries in L2 are created when embedded DTLS packets are received. Entries MAY be sent more than once to improve robustness against STUN loss.

4.2. Sending a STUN Binding Request or Response

When sending a STUN Binding Request or Response, the ICE agent MUST follow the steps below:

1. Embed any pending ACKs from L2 in a DTLS-IN-STUN-ACK attribute.

2. If there is a pending DTLS handshake packet in L1 and sufficient space remains in the STUN message, embed one DTLS handshake packet from L1 into a DTLS-IN-STUN-DATA attribute. When multiple packets are pending in L1, round-robin selection is RECOMMENDED.

3. Otherwise, include DTLS-IN-STUN-DATA with an empty value simply to indicate SPED support.

4.3. Receiving a STUN Binding Request or Response

When receiving a STUN Binding Request or Response, the ICE agent MUST follow the steps below:

1. If this is the first authenticated STUN message received from the peer, and the DTLS-IN-STUN-DATA attribute is not present, conclude that the peer does not support SPED, and conclude SPED processing.

2. If the STUN message contains a DTLS-IN-STUN-ACK attribute, process the CRC-32 values in the attribute and remove each ACKed DTLS handshake packet from L1.

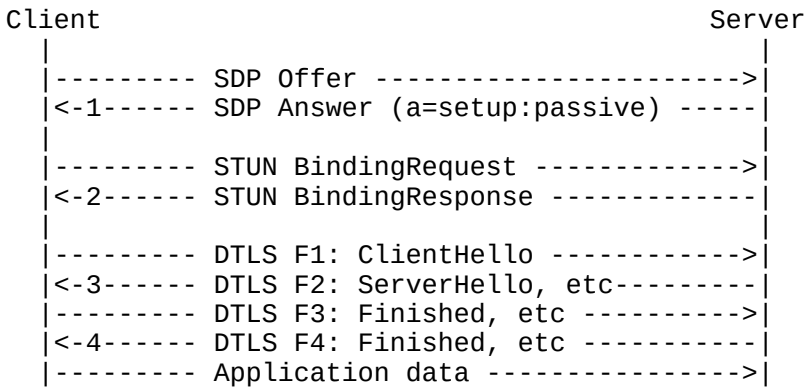
3. If the STUN message contains a non-empty DTLS-IN-STUN-DATA attribute, inject the attribute data into the DTLS layer and add the CRC-32 of the attribute value to L2.

4.4. Termination

Once a valid ICE candidate pair exists and direct sending is possible, implementations MAY terminate use of SPED and send DTLS directly. Implementations MAY instead continue to send embedded DTLS until DTLS handshaking is complete, for example, to continue to use SPED’s explicit acknowledgement mechanism.

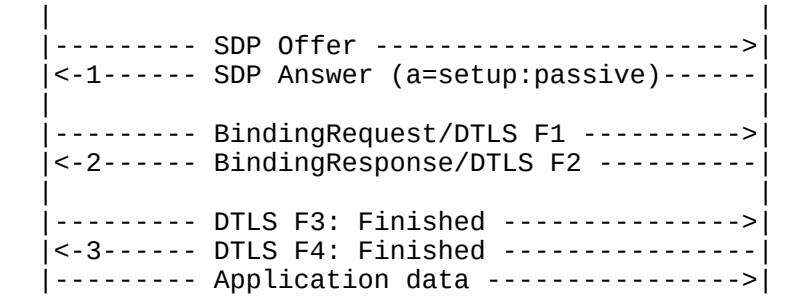
5. Examples

5.1. Vanilla DTLS 1.2

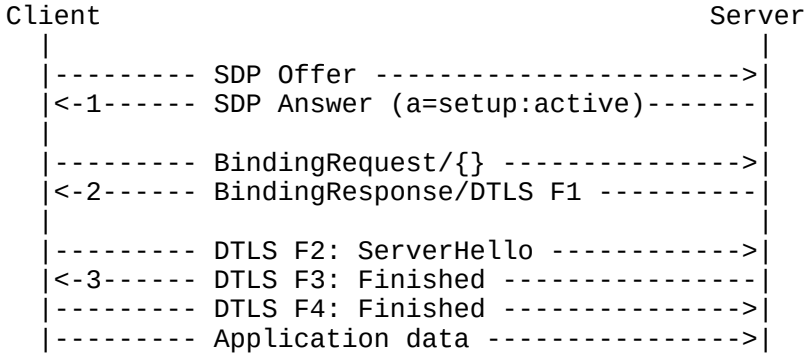


5.2. DTLS 1.2 with SPED

With a=setup:passive in the SDP answer, the offerer is the DTLS client:

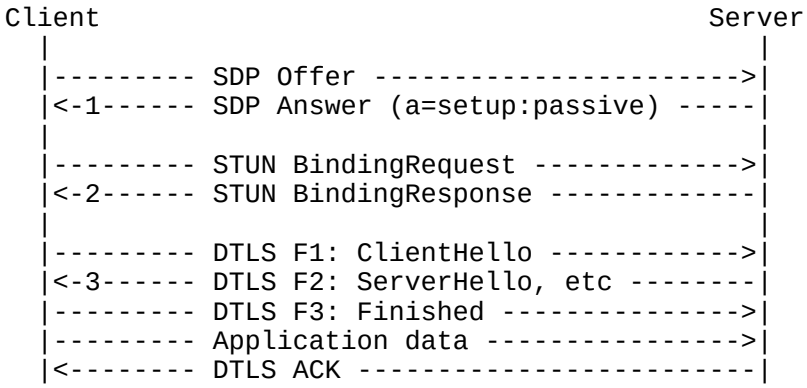


With a=setup:active in the SDP answer, the answerer is the DTLS client:



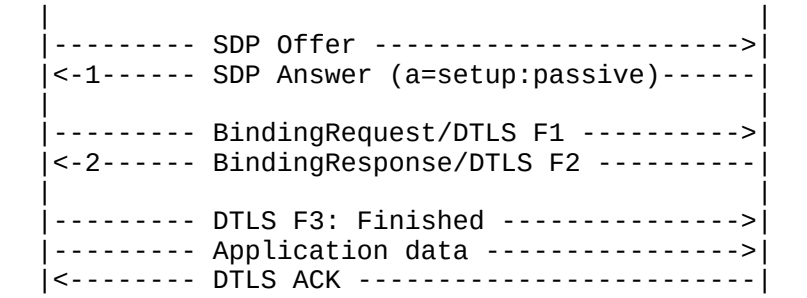
The flows are similar when the server uses ICE Lite.

5.3. Vanilla DTLS 1.3

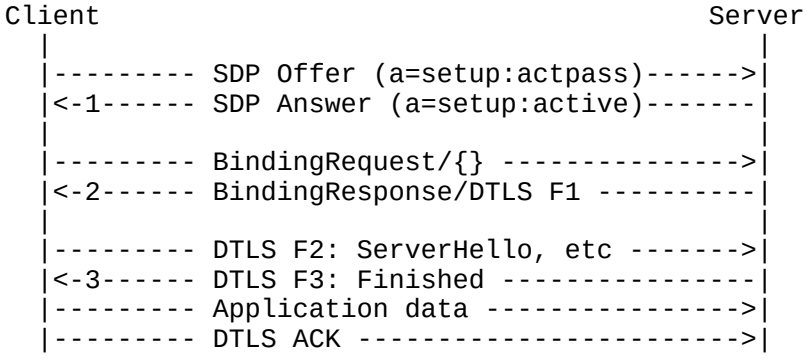


5.4. DTLS 1.3 with SPED

With a=setup:passive in the SDP answer, the offerer is the DTLS client:

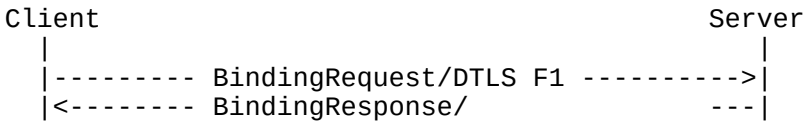


With a=setup:active in the SDP answer, the answerer is the DTLS client, and an additional RTT is incurred:

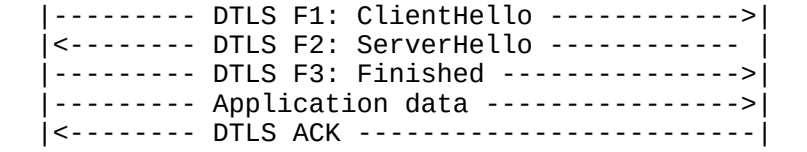


Again, the flows are similar when the server uses ICE Lite.

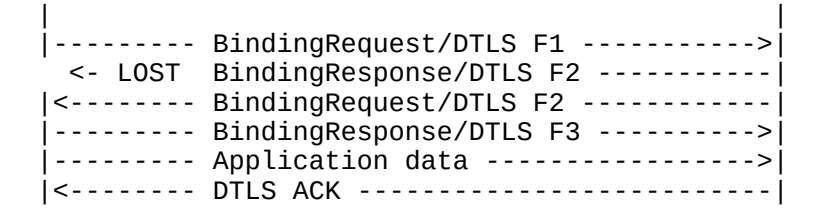
5.5. DTLS 1.3 with Non-SPED Peer



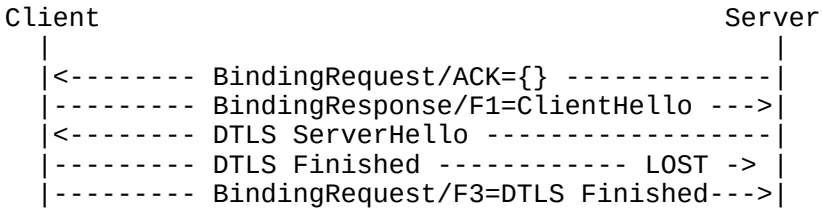
The absence of a DTLS-IN-STUN-DATA attribute allows the client to conclude that the server does not support SPED.



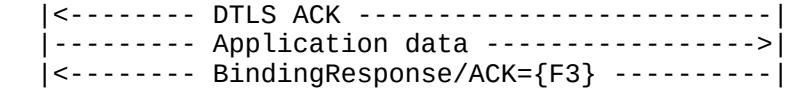
5.6. DTLS 1.3 with Flight 2 Loss



5.7. DTLS 1.3 with Flight 3 Loss

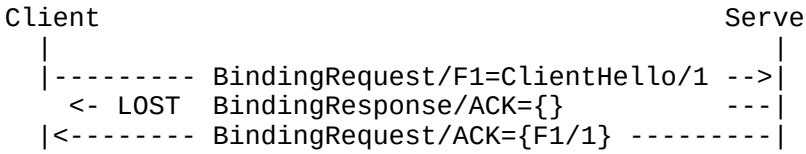


Note: The embedding continues until both client and server are known to be writable, but ICE does not send any packets it would not otherwise send.

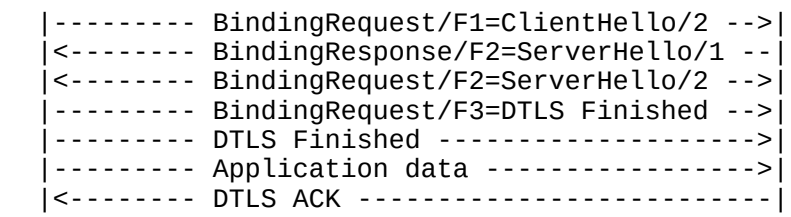


5.8. DTLS 1.3 PQC with Certificate Fragment Loss

The DTLS ClientHello is split into 2 packets. The DTLS ServerHello is split into 2 packets.

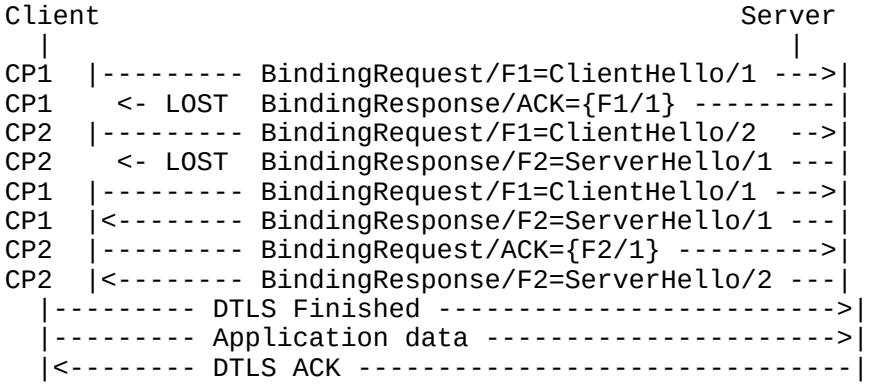


Note: When the server sends ACK{F1/1}, it does not yet have a DTLS packet to send since both of the packets from the ClientHello have arrived.



5.9. DTLS 1.3 PQC with Multiple Candidate Pairs and Certificate Fragment Loss

The DTLS ClientHello is split into 2 packets. The DTLS ServerHello is split into 2 packets. There are two candidate pairs, CP1 and CP2. The ICE agent retransmits BindingRequest once.



6. Implementation Notes

The following configuration for the SPED stack is RECOMMENDED. Note that this guidance may change based on implementation and deployment experience:

1. When SPED is active, disable internal DTLS timeouts, and resume them when receiving the first STUN Binding Response.

2. When using a PQC cipher suite, reduce the DTLS MTU as needed so embedded DTLS packets still fit within the expected path MTU. Experiments with an MTU near 900 bytes have been promising, but the best fragmentation strategy requires more study.

7. Prior Work

7.1. ICE-DTLS

The [I-D.thomson-rtcweb-ice-dtls] draft from 2012 proposed a similar mechanism to SPED, in which a single RTT could be removed from session setup by replacing STUN Request and Response messages with DTLS ClientHello and ServerHello messages, rather than piggybacking the DTLS messages as SPED does.

The ICE-DTLS mechanism ends up being considerably more complex than SPED, on account of the fact that the entirety of ICE functionality needs to be ported over to DTLS, for example consent checks, or retained as a complementary approach, for example peer address discovery. Furthermore, since it changes the details of connectivity negotiation, it is not backward compatible and therefore must be negotiated via SDP ice-options.

8. Future Work

Embedding data into STUN requests is a technique that could also be used for early transmission or improved reliability of other important data. For example, one could imagine transferring key-frames, if small enough, or RTCP or SCTP control messages. However, this has not been thoroughly sketched out in this proposal.

9. Security Considerations

9.1. DTLS Replay and Spoofing

This specification uses application-layer caching of DTLS packets which means packets can be sent multiple times using the same sequence number. For the receiver these are considered replays if received multiple times and rejected as described in Section 4.5.1 of [RFC9147].

The embedded DTLS handshake is authenticated by existing ICE logic, i.e. the ICE USERNAME and MESSAGE-INTEGRITY mechanisms. Any spoofed ICE packets are rejected accordingly.

9.2. Pacing and Congestion

The protocol defined in this specification increases the size of the STUN packets that are sent by the ICE agent to a peer without knowing if that peer can use the embedded data. Although the initial data sent is just the DTLS ClientHello, this packet can be close to a MTU when a PQC cipher suite is used. If this is unacceptable, an offer-answer mechanism for SPED can be used to address this concern.

The STUN requests used for embedding DTLS are already paced as described in Appendix B.1 of [RFC8445], which limits the outgoing bandwidth from this mechanism. However, that pacing assumes an ICE check of "less than 120 bytes", which will not be the case when a DTLS ClientHello is embedded, especially a PQC one.

Solutions to this problem require transmitting the DTLS ClientHello less often, perhaps only on certain candidate pairs, and is a subject for further study.

10. IANA Considerations

This document defines two new STUN attributes, DTLS-IN-STUN-DATA and DTLS-IN-STUN-ACK. These attributes need to be registered with IANA in the "STUN Attributes" registry, following the procedures defined in [RFC8489]. Provisional names have been used in this draft and the registry.

11. References

11.1. 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/rfc/rfc2119>.

[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/rfc/rfc8174>.

11.2. Informative References

[I-D.thomson-rtcweb-ice-dtls] Thomson, M., "Using Datagram Transport Layer Security (DTLS) For Interactivity Connectivity Establishment (ICE) Connectivity Checking: ICE-DTLS", Work in Progress, Internet-Draft, draft-thomson-rtcweb-ice-dtls-00, 27 March 2012, <https://datatracker.ietf.org/doc/html/draft-thomson-rtcweb-ice-dtls-00>.

[RFC4568] Andreasen, F., Baugher, M., and D. Wing, "Session Description Protocol (SDP) Security Descriptions for Media Streams", RFC 4568, DOI 10.17487/RFC4568, July 2006, <https://www.rfc-editor.org/rfc/rfc4568>.

[RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, DOI 10.17487/RFC5245, April 2010, <https://www.rfc-editor.org/rfc/rfc5245>.

[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/rfc/rfc5389>.

[RFC5763] Fischl, J., Tschofenig, H., and E. Rescorla, "Framework for Establishing a Secure Real-time Transport Protocol (SRTP) Security Context Using Datagram Transport Layer Security (DTLS)", RFC 5763, DOI 10.17487/RFC5763, May 2010, <https://www.rfc-editor.org/rfc/rfc5763>.

[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/rfc/rfc5764>.

[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/rfc/rfc6347>.

[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/rfc/rfc8445>.

[RFC8489] Petit-Huguenin, M., Salgueiro, G., Rosenberg, J., Wing, D., Mahy, R., and P. Matthews, "Session Traversal Utilities for NAT (STUN)", RFC 8489, DOI 10.17487/RFC8489, February 2020, <https://www.rfc-editor.org/rfc/rfc8489>.

[RFC8656] Reddy, T., Ed., Johnston, A., Ed., Matthews, P., and J. Rosenberg, "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)", RFC 8656, DOI 10.17487/RFC8656, February 2020, <https://www.rfc-editor.org/rfc/rfc8656>.

[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/rfc/rfc8829>.

[RFC8831] Jesup, R., Loreto, S., and M. T端xen, "WebRTC Data Channels", RFC 8831, DOI 10.17487/RFC8831, January 2021, <https://www.rfc-editor.org/rfc/rfc8831>.

[RFC9147] Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", RFC 9147, DOI 10.17487/RFC9147, April 2022, <https://www.rfc-editor.org/rfc/rfc9147>.

[RFC9443] Aboba, B., Salgueiro, G., and C. Perkins, "Multiplexing Scheme Updates for QUIC", RFC 9443, DOI 10.17487/RFC9443, July 2023, <https://www.rfc-editor.org/rfc/rfc9443>.

Appendix A. Benchmark Numbers

For the scenario without packet loss, benchmarking is straightforward, and the savings from SPED amount to 1 RTT, as expected. However, in packet loss scenarios, the savings can be much larger, especially in the worst (p95) cases. This is likely due in part to using ICE pacing rather than exponential backoff for DTLS retransmissions.

In this benchmark, a 200 ms RTT is used. Packet loss is simulated using the virtual network mechanism in Google’s libwebrtc. Duration is measured as time from start until both peers have completed the DTLS handshake.

A.1. DTLS 1.3 with PQC

DTLS 1.3 with PQC	Loss %	p10 (ms)	p50	average	p95
Vanilla	0	850	850	850	850
SPED	0	650	650	650	650
Vanilla	5%	850	850	947	1253
SPED	5%	650	650	656	700
Vanilla	10%	850	900	1193	2170
SPED	10%	650	650	685	800
Vanilla	25%	850	1350	2020	3080
SPED	25%	650	750	850	1105

Table 2

A.2. DTLS 1.3

DTLS 1.3	Loss %	p10 (ms)	p50	average	p95
Vanilla	0	750	750	750	750
SPED	0	550	550	550	550
Vanilla	5%	750	750	800	1150
SPED	5%	550	550	555	600
Vanilla	10%	750	750	935	1200
SPED	10%	550	550	560	600
Vanilla	25%	750	1150	1400	2560

SPED	25%	550	600	620	750

Table 3

A.3. DTLS 1.2

DTLS 1.2	Loss %	p10 (ms)	p50	average	p95
Vanilla	0	850	850	850	850
SPED	0	650	650	650	650
Vanilla	5%	850	850	916	1300
SPED	5%	650	650	695	1150
Vanilla	10%	850	900	1133	2150
SPED	10%	650	650	690	760
Vanilla	25%	850	1350	3920	8860
SPED	25%	750	750	862	1400

Table 4

Acknowledgments

Authors' Addresses

Philipp Hancke
Meta Platforms Inc.
Email: philipp.hancke@googlemail.com

Justin Uberti
OpenAI
Email: justin@uberti.name

Jonas Oreland
Google
Email: jonaso@google.com