Abstract

This document describes these new SDP "proto" attribute values: "QUIC", "QUIC/RTP/SAVP", "QUIC/RTP/AVPF", and "QUIC/RTP/SAVPF", and describes how SDP Offer/Answer can be used to set up an RTP connection using QUIC as a transport protocol.

These proto values are necessary to allow the use of QUIC as an underlying transport protocol for applications such as SIP and WebRTC that commonly use SDP as a session signaling protocol to set up RTP connections.

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/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 1 August 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.
1. Introduction

This document describes these new SDP "proto" attribute values: "QUIC", "QUIC/RTP/SAVP", "QUIC/RTP/AVPF", and "QUIC/RTP/SAVPF", and describes how SDP Offer/Answer ([RFC3264]) can be used to set up an RTP ([RFC3550]) connection using QUIC ([RFC9000] and related specifications) as a transport protocol.

These proto values are necessary to allow the use of QUIC as an underlying transport protocol for applications such as SIP ([RFC3264]) and WebRTC ([RFC8825]) that commonly use SDP as a session signaling protocol to set up RTP connections.

1.1. Notes for Readers

(Note to RFC Editor - if this document ever reaches you, please remove this section)
This document is intended for publication as a standards-track RFC in the IETF stream, but has not been adopted by any IETF working group, and does not carry any special status within the IETF.

1.2. Terminology

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.

1.3. Scope of this document

This document focuses on the IANA registration and description of the RTP sessions using SDP Offer/Answer, as would be the case for many current RTP applications in common use, such as SIP ([RFC3261]) and WebRTC ([RFC8825]).

This document is intended as complementary to drafts such as [I-D.engelbart-rtp-over-quic], which largely focus on RTP/RTCP encapsulation in QUIC, so that the SDP experts can focus on SDP offer/answer aspects, and the RTP experts can focus on RTP/RTCP encapsulation aspects.

1.4. Contribution and Discussion Venues for this draft.

(Note to RFC Editor - if this document ever reaches you, please remove this section)

With the concurrence of the AVTCORE and MMUSIC working group co-chairs, this document should be discussed in the AVTCORE working group, in the same venue where RTP over QUIC proposals are being discussed. When proposals for RTP over SIP have stabilized in AVTCORE, this document will be sent to the MMUSIC working group for review by SDP experts, but SDP-specific comments are welcomed at any time.

Readers are also invited to open issues and send pull requests with contributed text for this document in the GitHub repository at https://github.com/SpencerDawkins/sdp-rtp-quic. The direct link to the list of issues is https://github.com/SpencerDawkins/sdp-rtp-quic/issues.
1.5. Assumptions for this document

This document assumes that for RTP-over-QUIC, it is useful to register these AVP profiles using QUIC, in order to allow existing SIP and RTCWEB RTP applications to migrate more easily to QUIC:

* RTP/SAVP ("The Secure Real-time Transport Protocol (SRTP)")], as defined in [RFC3711].

* RTP/AVPF ("Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/AVPF)"), as defined in [RFC4585].

* RTP/SAVPF ("Extended Secure RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/SAVPF)"), as defined in [RFC5124].

This document assumes that any implementation adding support for RTP-over-QUIC could reasonably also add support for BUNDLE ([RFC8843]) and "rtcp-mux" ([RFC5761]), so these capabilities are not mentioned further in this document.

1.5.1. An Aside on Secure AVP Profiles in an RTP Over QUIC Context

Existing RTP implementations have the choice for any given RTP connection to exchange either unencrypted RTP streams (using AVP profiles such as RTP/AVPF) or encrypted RTP streams (using AVP profiles such as RTP/SAVPF).

An RTP implementation that uses QUIC as its underlying transport protocol will always send an RTP stream that is encrypted between the two QUIC endpoints, so some RTP implementations may be tempted to exchange unencrypted RTP as an encrypted QUIC payload, reasoning that QUIC protection will be sufficient.

One nuance here is that QUIC is heavily encrypted between two QUIC endpoints, with the very minimal exception of the invariant header fields described in [RFC8999], but as described in [RFC7667], many RTP applications use middleboxes for a variety of reasons, and some of these topologies (for example, media translation) require that the middlebox understand the RTP payload.

These middleboxes are explicitly addressed, and the QUIC cryptographic handshake described in [RFC9001] takes place between the RTP endpoint and the RTP middlebox. After the QUIC cryptographic handshake has succeeded, the RTP middlebox has access to the RTP in the QUIC payload, and can perform whatever translations are appropriate before forwarding the RTP steam to another RTP endpoint. However, if the
RTP sender uses one of the "insecure" AVPs, the middlebox does not have any indication that the RTP sender wants the translated RTP stream to be protected by encryption when the middlebox forwards it. That might be fine if the middlebox and RTP endpoint are both using RTP over QUIC, but if the middlebox is performing transport translation as well, the middlebox may also be translating an RTP-over-QUIC stream to RTP-over-UDP.

This specification tries to provide that indication by supporting both "secure" and "insecure" AVPs for RTP over QUIC, so the middlebox that is providing back-to-back RTP sessions as described in [RFC7667] can be aware of the sender’s desire that a translated RTP stream is encrypted regardless of the underlying transport protocol, without always requiring both SRTP and QUIC encryption between each pair of QUIC endpoints for all RTP traffic. That’s one strategy, and it’s certainly possible that other strategies might be safer, cleaner, and/or more useful.

1.6. Open Questions

The current contents of Section 2 and Section 3 would allow an existing RTP/RTCP implementation to make a relatively straightforward transition from "RTP over UDP" to "RTP over QUIC datagrams over UDP", and likewise from "RTCP over UDP" to "RTCP over QUIC datagrams over UDP".

Although it is still early days for RTP over QUIC, things may not be that straightforward. Just limiting our attention to various proposals for "RTP over QUIC" that have already been discussed on the Media Over QUIC IETF mailing list [MOQ] and in various IETF side meetings, we have seen

* a desire to make use of QUIC connection migration in case of path failure between two endpoints

* a desire to replace RTP Round Trip Time (RTT) measurement with something like a proposed QUIC extension for timestamps ([I-D.huitema-quic-ts]) that could be used to measure one-way delays

* a desire to make use of QUIC streams, potentially with QUIC datagrams in the same QUIC connection

* a desire to decouple the RTP state machine and the QUIC state machine, which currently assume they are solely responsible for managing sending rates, without any knowledge of what the other plans to do
* a desire to select a media-focused congestion control mechanism such as "Self-Clocked Rate Adaptation for Multimedia", or SCReAM ([RFC8298]), that can be included in QUIC implementations

* a desire to use RTP over QUIC in peer-to-peer applications, which likely would require extensions to the QUIC protocol for NAT traversal, at a bare minimum

Changes to the SDP signaling in Section 2 and Section 3 may be (and likely would be) needed in order to support any of these desires (or other desires that may surface in the future).

2. Identifiers and Attributes

As much as possible, these are reused from other specifications, with references to the original definitions.

2.1. Protocol Identifiers

2.1.1. The QUIC proto

The ‘QUIC’ protocol identifier is similar to the ‘UDP’ and ‘TCP’ protocol identifiers in that it only describes the transport protocol, and not the upper-layer protocol.

An ‘m’ line that specifies ‘QUIC’ MUST further qualify the application-layer protocol using an fmt identifier, such as "QUIC/RTP/AVPF". Media described using an ‘m’ line containing the ‘QUIC’ protocol identifier are carried using QUIC ([RFC9000]).

The following is an update to the ABNF for an ‘m’ line, as specified by [RFC8866], that defines a new value for the QUIC protocol.

```
media-field =         %s"m" "=" media SP port \["/" integer\] SP proto 1*(SP fmt) CRLF
m= line parameter        parameter value(s)
```

2.1.2. The QUIC/RTP/SAVP proto

The following is an update to the ABNF for an ‘m’ line, as specified by [RFC8866], that defines a new value for the QUIC/RTP/SAVP protocol.
media-field = %s"m" =" media SP port \["/" integer\]
              SP proto 1*(SP fmt) CRLF

m= line parameter          parameter value(s)
------------------------------------------------------------------
<media>:                 (unchanged from {{RFC8866}})
<proto>:                 'QUIC/RTP/SAVPF'
<port>:                  UDP port number
<fmt>:                   (unchanged from {{RFC8866}})

2.1.3. The QUIC/RTP/AVPF proto

The following is an update to the ABNF for an 'm' line, as specified
by [RFC8866], that defines a new value for the QUIC/RTP/AVPF
protocol.

media-field = %s"m" =" media SP port \["/" integer\]
              SP proto 1*(SP fmt) CRLF

m= line parameter          parameter value(s)
------------------------------------------------------------------
<media>:                 (unchanged from {{RFC8866}})
<proto>:                 'QUIC/RTP/AVPF'
<port>:                  UDP port number
<fmt>:                   (unchanged from {{RFC8866}})

2.1.4. The QUIC/RTP/SAVPF proto

The following is an update to the ABNF for an 'm' line, as specified
by [RFC8866], that defines a new value for the QUIC/RTP/SAVPF
protocol.

media-field = %s"m" =" media SP port \["/" integer\]
              SP proto 1*(SP fmt) CRLF

m= line parameter          parameter value(s)
------------------------------------------------------------------
<media>:                 (unchanged from {{RFC8866}})
<proto>:                 'QUIC/RTP/SAVPF'
<port>:                  UDP port number
<fmt>:                   (unchanged from {{RFC8866}})

2.2. A QUIC/RTP/AVPF Offer

A complete example of an SDP offer using QUIC/RTP/AVPF might look
like:

Dawkins                   Expires 1 August 2022                 [Page 7]
<table>
<thead>
<tr>
<th>SDP line</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>v=0</td>
<td>Same as [RFC8866]</td>
</tr>
<tr>
<td>o=jdoe 3724394400 3724394405 IN IP4 198.51.100.1</td>
<td>Same as [RFC8866]</td>
</tr>
<tr>
<td>s=Call to John Smith</td>
<td>Same as [RFC8866]</td>
</tr>
<tr>
<td>i=SDP Offer #1</td>
<td>Same as [RFC8866]</td>
</tr>
<tr>
<td>u=<a href="http://www.jdoe.example.com/home.html">http://www.jdoe.example.com/home.html</a></td>
<td>Same as [RFC8866]</td>
</tr>
</tbody>
</table>
| e=Jane Doe 
  jane@jdoe.example.com 
  (mailto:jane@jdoe.example.com) | Same as [RFC8866] |
| p=+1 617 555-6011 | Same as [RFC8866] |
| c=IN IP4 198.51.100.1 | Same as [RFC8866] |
| t=0 0 | Same as [RFC8866] |
| m=audio 49170 RTP/AVP 0 | Same as [RFC8866] |
| m=audio 49180 RTP/AVP 0 | Same as [RFC8866] |
| m=video 51372 QUIC/RTP/AVPF 99 | QUIC transport |
| a=setup:passive | will wait for QUIC handshake (setup attribute from [RFC4145]) |
| a=connection:new | don’t want to reuse an existing QUIC connection (connection attribute from [RFC4145]) |
| c=IN IP6 2001:db8::2 | Same as [RFC8866] |
| a=rtpmap:99 h266/90000 | H.266 VVC codec [I-D.ietf-avtcore-rtp-vvc] |

Table 1
This example is largely based on an example appearing in [RFC8866], Section 5, but is using QUIC/RTP/AVPF to support a newer codec.

Because QUIC uses connections for both streams and datagrams, we are reusing two session- and media-level SDP attributes from [SDP-attribute-name] that were defined in [RFC4145] for use with TCP: setup and connection.

This example SDP offer might be included in a SIP Invite.

3. IANA Considerations

This document registers these protocols in the proto registry ([SDP-parameters]).

* QUIC (Section 2.1.1)
* QUIC/RTP/SAVP (Section 2.1.2)
* QUIC/RTP/AVPF (Section 2.1.3)
* QUIC/RTP/SAVPF (Section 2.1.4)

3.1. Proto Registrations

IANA is requested to add these protocols to the Session Description Protocol (SDP) Parameters proto registry ([SDP-parameters]).

| Type | SDP Name       | Reference |
|------|----------------+-----------|
| proto| QUIC           | RFCXXXX   |
| proto| QUIC/RTP/SAVP  | RFCXXXX   |
| proto| QUIC/RTP/AVPF  | RFCXXXX   |
| proto| QUIC/RTP/SAVPF | RFCXXXX   |

Table 2

*Note to the RFC Editor*

Please replace "RFCXXXX" with the assigned RFC number, when that is available, and remove this note.
4. Security Considerations

Security considerations for the QUIC protocol are described in the corresponding section in [RFC9000].

Security considerations for the TLS handshake used to secure QUIC are described in [RFC9001].

Security considerations for SDP are described in the corresponding section in [RFC8866].

Security considerations for SDP offer/answer are described in the corresponding section in [RFC3264].

5. Acknowledgments

My appreciation to the authors of [RFC4145], which served as a model for the initial structure of this document.

Thanks to these folks for helping to improve this draft:

* Colin Perkins

(Your name also could appear here. Please comment and contribute, as per Section 1.4).

6. References

6.1. Normative References


6.2. Informative References


[I-D.ietf-avtcore-rtp-vvc] Zhao, S., Wenger, S., Sanchez, Y., Wang, Y., and M. M. Hannuksela, "RTP Payload Format for Versatile Video Coding (VVC)", Work in Progress, Internet-Draft, draft-ietf-
avtcore-rtp-vvc-13, 18 November 2021,

[RFC4145] Yon, D. and G. Camarillo, "TCP-Based Media Transport in
the Session Description Protocol (SDP)", RFC 4145,
DOI 10.17487/RFC4145, September 2005,

Author’s Address

Spencer Dawkins
Tencent America LLC
United States of America

Email: spencerdawkins.ietf@gmail.com
Abstract

This document describes these new SDP "proto" attribute values: "QUIC", "QUIC/RTP/SAVP", "QUIC/RTP/AVPF", and "QUIC/RTP/SAVPF", and describes how SDP Offer/Answer can be used to set up an RTP connection using QUIC as a transport protocol.

These proto values are necessary to allow the use of QUIC as an underlying transport protocol for applications that commonly use SDP as a session signaling protocol to set up RTP connections with UDP as its underlying transport protocol, such as SIP and WebRTC.

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/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 12 March 2022.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.
1. Introduction

This document describes these new SDP "proto" attribute values: "QUIC", "QUIC/RTP/SAVP", "QUIC/RTP/AVPF", and "QUIC/RTP/SAVPF", and describes how SDP Offer/Answer ([RFC3264]) can be used to set up an RTP ([RFC3550]) connection using QUIC ([RFC9000]) as a transport protocol.

These proto values are necessary to allow the use of QUIC as an underlying transport protocol for applications that commonly use SDP as a session signaling protocol to set up RTP connections with UDP as its underlying transport protocol, such as SIP ([RFC3261]) and WebRTC ([RFC8825]).
1.1. Notes for Readers

This document is intended for publication as a standards-track RFC in the IETF stream, but has not been adopted by any IETF working group, and does not carry any special status within the IETF.

1.2. Terminology

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.

1.3. Contribution and Discussion Venues for this draft.

(Note to RFC Editor - if this document ever reaches you, please remove this section)

This document is under development in the Github repository at https://github.com/SpencerDawkins/sdp-rtp-quic.

Readers are invited to open issues and send pull requests with contributed text for this document, or to send them to the author via email.

1.4. Scope of this document

This document focuses on the IANA registration and description of the RTP sessions using SDP Offer/Answer, as would be the case for many current RTP applications in common use, such as SIP ([RFC3261]) and WebRTC ([RFC8225]).

This document is intended as complementary to [I-D.engelbart-rtp-over-quic], which largely focuses on RTP/RTCP encapsulation in QUIC datagrams, so that the SDP experts can focus on SDP offer/answer aspects, and the RTP experts can focus on RTP/RTCP encapsulation aspects.

1.5. Assumptions for this document

This document assumes that for RTP-over-QUIC, it is useful to register these AVP profiles using QUIC, in order to allow existing SIP and RTCWEB RTP applications to migrate more easily to QUIC:

* RTP/SAVP ("The Secure Real-time Transport Protocol (SRTP)"), as defined in [RFC3711].
* RTP/AVPF ("Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/AVPF)"), as defined in [RFC4585].

* RTP/SAVPF ("Extended Secure RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/SAVPF)"), as defined in [RFC5124].

This document assumes that any implementation adding support for RTP-over-QUIC could reasonably add support for BUNDLE ([RFC8843]), "rtcp-mux" ([RFC5761]).

2. Open Questions (probably not for this draft, but could have implications on SDP Offer/Answer)

* RTP (and RTCP) headers and payloads will be entirely encrypted using QUIC ([RFC9000]), as secured by TLS 1.3 handshake ([RFC9001]), between QUIC endpoints. It’s worth thinking more about how that maps onto expected deployment scenarios like centralized multiparty conferencing, and also whether WebRTC really requires SAVPF with double encryption (i.e. SRTP encryption, and then QUIC encryption). No opinions here yet, just noting the question for now.

* When QUIC establishes connections, it uses IP addresses but then expects applications to use connection IDs to refer to connections, even if the underlying IP addresses change because of NAT binding, and even if the QUIC implementation performs QUIC connection migration itself, so the underlying IP addresses change. RTP applications expect to use IP addresses, not QUIC connection IDs. Must we specify an RTP/RTCP adaptation layer, similar to [I-D.ietf-quic-http] for HTTP/3?

3. Identifiers and Attributes

As much as possible, these are reused from other specifications, with references to the original definitions.

3.1. Protocol Identifiers

3.1.1. The QUIC proto

The 'QUIC' protocol identifier is similar to the 'UDP' and 'TCP' protocol identifiers in that it only describes the transport protocol, and not the upper-layer protocol.
An ‘m’ line that specifies ‘QUIC’ MUST further qualify the application-layer protocol using an fmt identifier, such as "QUIC/RTP/AVPF". Media described using an ‘m’ line containing the ‘QUIC’ protocol identifier are carried using QUIC ([RFC9000]).

The following is an update to the ABNF for an ‘m’ line, as specified by [RFC8866], that defines a new value for the QUIC protocol.

```
media-field = %s"m" "=" media SP port \["/" integer\] SP proto 1*(SP fmt) CRLF

m= line parameter parameter value(s)
```

<table>
<thead>
<tr>
<th>parameter</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>media</td>
<td>(unchanged from {{RFC8866}})</td>
</tr>
<tr>
<td>proto</td>
<td>'QUIC'</td>
</tr>
<tr>
<td>port</td>
<td>UDP port number</td>
</tr>
<tr>
<td>fmt</td>
<td>(unchanged from {{RFC8866}})</td>
</tr>
</tbody>
</table>

3.1.2. The QUIC/RTP/SAVP proto

The following is an update to the ABNF for an ‘m’ line, as specified by [RFC8866], that defines a new value for the QUIC/RTP/SAVP protocol.

```
media-field = %s"m" "=" media SP port \["/" integer\] SP proto 1*(SP fmt) CRLF

m= line parameter parameter value(s)
```

<table>
<thead>
<tr>
<th>parameter</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>media</td>
<td>(unchanged from {{RFC8866}})</td>
</tr>
<tr>
<td>proto</td>
<td>'QUIC/RTP/SAVP'</td>
</tr>
<tr>
<td>port</td>
<td>UDP port number</td>
</tr>
<tr>
<td>fmt</td>
<td>(unchanged from {{RFC8866}})</td>
</tr>
</tbody>
</table>

3.1.3. The QUIC/RTP/AVPF proto

The following is an update to the ABNF for an ‘m’ line, as specified by [RFC8866], that defines a new value for the QUIC/RTP/AVPF protocol.
3.1.4. The QUIC/RTP/SAVPF proto

The following is an update to the ABNF for an 'm' line, as specified by [RFC8866], that defines a new value for the QUIC/RTP/SAVPF protocol.

media-field = %s"m" "=" media SP port \["/" integer\] \SP proto 1*(SP fmt) CRLF

m= line parameter parameter value(s)

------------------------------------------------------------------

<media>: (unchanged from {{RFC8866}})
<proto>: 'QUIC/RTP/SAVPF'
<port>: UDP port number
<fmt>: (unchanged from {{RFC8866}})

3.2. A QUIC/RTP/AVPF Offer

A complete example of an SDP offer using QUIC/RTP/AVPF might look like:
+--------------------------------+=================================+
| SDP line                       | Notes                           |
+================================+=================================+
| v=0                            | Same as [RFC8866]               |
+--------------------------------+---------------------------------+
| o=jdoe 3724394400 3724394405   | Same as [RFC8866]               |
| IN IP4 198.51.100.1            |                                 |
+--------------------------------+---------------------------------+
| s=Call to John Smith           | Same as [RFC8866]               |
+--------------------------------+---------------------------------+
| i=SDP Offer #1                 | Same as [RFC8866]               |
+--------------------------------+---------------------------------+
| u=http://www.jdoe.example.com/ | Same as [RFC8866]               |
| home.html                      |                                 |
+--------------------------------+---------------------------------+
| e=Jane Doe                     | Same as [RFC8866]               |
| jane@jdoe.example.com          |                                 |
| (mailto:jane@jdoe.example.com)  |                                 |
+--------------------------------+---------------------------------+
| p=+1 617 555-6011              | Same as [RFC8866]               |
+--------------------------------+---------------------------------+
| c=IN IP4 198.51.100.1          | Same as [RFC8866]               |
+--------------------------------+---------------------------------+
| t=0 0                          | Same as [RFC8866]               |
+--------------------------------+---------------------------------+
| m=audio 49170 RTP/AVP 0        | Same as [RFC8866]               |
+--------------------------------+---------------------------------+
| m=audio 49180 RTP/AVP 0        | Same as [RFC8866]               |
+--------------------------------+---------------------------------+
| m=video 51372 QUIC/RTP/AVPF 99 | QUIC transport                  |
+--------------------------------+---------------------------------+
| a=setup:passive                | will wait for QUIC handshake    |
|                                | (setup attribute from [RFC4145])|
+--------------------------------+---------------------------------+
| a=connection:new               | don’t want to reuse an existing|
|                                | QUIC connection (connection    |
|                                | attribute from [RFC4145])       |
+--------------------------------+---------------------------------+
| c=IN IP6 2001:db8::2           | Same as [RFC8866]               |
+--------------------------------+---------------------------------+
| a=rtpmap:99 h266/90000 H.266 VVC codec |
|                                | [I-D.ietf-avtcore-rtp-vvc]      |
+--------------------------------+---------------------------------+

Table 1
This example is largely based on an example appearing in [RFC8866], Section 5, but is using QUIC/RTP/AVPF to support a newer codec.

Because QUIC uses connections for both streams and datagrams, we are reusing two session- and media-level SDP attributes from [SDP-attribute-name] that were defined in [RFC4145] for use with TCP: setup and connection.

This example SDP offer might be included in a SIP Invite.

4.  IANA Considerations

This document registers these protocols in the proto registry ([SDP-parameters]).

* QUIC (Section 3.1.1)
* QUIC/RTP/SAVP (Section 3.1.2)
* QUIC/RTP/AVPF (Section 3.1.3)
* QUIC/RTP/SAVPF (Section 3.1.4)

4.1.  Proto Registrations

IANA is requested to add these protocols to the Session Description Protocol (SDP) Parameters proto registry ([SDP-parameters]).

<table>
<thead>
<tr>
<th>Type</th>
<th>SDP Name</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>proto</td>
<td>QUIC</td>
<td>RFCXXXX</td>
</tr>
<tr>
<td>proto</td>
<td>QUIC/RTP/SAVP</td>
<td>RFCXXXX</td>
</tr>
<tr>
<td>proto</td>
<td>QUIC/RTP/AVPF</td>
<td>RFCXXXX</td>
</tr>
<tr>
<td>proto</td>
<td>QUIC/RTP/SAVPF</td>
<td>RFCXXXX</td>
</tr>
</tbody>
</table>

Table 2

*Note to the RFC Editor*

Please replace "RFCXXXX" with the assigned RFC number, when that is available, and remove this note.
5. Security Considerations

Security considerations for the QUIC protocol are described in the corresponding section in [RFC9000].

Security considerations for the TLS handshake used to secure QUIC are described in [RFC9001].

Security considerations for SDP are described in the corresponding section in [RFC8866].

Security considerations for SDP offer/answer are described in the corresponding section in [RFC3264].

6. Acknowledgments

My appreciation to the authors of [RFC4145], which served as a model for the initial structure of this document.

Your name could appear here. Please comment and contribute, as per Section 1.3.

7. References

7.1. Normative References


8.2. Informative References

[I-D.engelbart-rtp-over-quic]

[I-D.ietf-avtcore-rtp-vvc]

[I-D.ietf-quic-http]


Author's Address

Spencer Dawkins
Tencent America LLC
United States of America

Email: spencerdawkins.ietf@gmail.com
Abstract

While the Secure Real-time Transport Protocol (SRTP) provides confidentiality for the contents of a media packet, a significant amount of metadata is left unprotected, including RTP header extensions and contributing sources (CSRCs). However, this data can be moderately sensitive in many applications. While there have been previous attempts to protect this data, they have had limited deployment, due to complexity as well as technical limitations.

This document defines Cryptex as a new mechanism that completely encrypts header extensions and CSRCs and uses simpler signaling with the goal of facilitating deployment.

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/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 8 September 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.
This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction .............................................. 3
   1.1. Problem Statement .................................... 3
   1.2. Previous Solutions .................................... 3
   1.3. Goals ................................................ 4
2. Terminology ................................................. 5
3. Design .................................................... 5
4. Signaling .................................................. 5
5. RTP Header Processing ....................................... 6
   5.1. Sending ................................................ 6
   5.2. Receiving .............................................. 7
6. Encryption and Decryption ................................. 7
   6.1. Packet Structure ..................................... 7
   6.2. Encryption Procedure ................................ 8
   6.3. Decryption Procedure ................................ 9
7. Backwards Compatibility .................................... 9
8. Security Considerations .................................. 9
9. IANA Considerations ....................................... 10
   9.1. cryptex SDP Attribute ................................ 10
10. Acknowledgements ........................................ 11
11. References ............................................... 11
   11.1. Normative References ................................ 11
   11.2. Informative References ............................... 12
Appendix A. Test Vectors .................................... 12
   A.1. AES-CTR .............................................. 12
      A.1.1. RTP Packet with 1-byte header extension ........ 12
      A.1.2. RTP Packet with 2-byte header extension ........ 13
      A.1.3. RTP Packet with 1-byte header extension and CSRC fields ............................................. 14
      A.1.4. RTP Packet with 2-byte header extension and CSRC fields ............................................. 15
      A.1.5. RTP Packet with empty 1-byte header extension and CSRC fields ......................................... 15
      A.1.6. RTP Packet with empty 2-byte header extension and CSRC fields ......................................... 16
   A.2. AES-GCM ............................................... 17
      A.2.1. RTP Packet with 1-byte header extension ........ 17
      A.2.2. RTP Packet with 2-byte header extension ........ 17
1.  Introduction

1.1.  Problem Statement

The Secure Real-time Transport Protocol [RFC3711] mechanism provides message authentication for the entire RTP packet, but only encrypts the RTP payload. This has not historically been a problem, as much of the information carried in the header has minimal sensitivity (e.g., RTP timestamp); in addition, certain fields need to remain as cleartext because they are used for key scheduling (e.g., RTP SSRC and sequence number).

However, as noted in [RFC6904], the security requirements can be different for information carried in RTP header extensions, including the per-packet sound levels defined in [RFC6464] and [RFC6465], which are specifically noted as being sensitive in the Security Considerations section of those RFCs.

In addition to the contents of the header extensions, there are now enough header extension identifiers in active use that the header extension identifiers themselves can provide meaningful information in terms of determining the identity of the endpoint and/or application. Accordingly, these identifiers can be considered a fingerprinting issue.

Finally, the CSRCs included in RTP packets can also be sensitive, potentially allowing a network eavesdropper to determine who was speaking and when during an otherwise secure conference call.

1.2.  Previous Solutions

[RFC6904] was proposed in 2013 as a solution to the problem of unprotected header extension values. However, it has not seen significant adoption, and has a few technical shortcomings.

First, the mechanism is complicated. Since it allows encryption to be negotiated on a per-extension basis, a fair amount of signaling logic is required. And in the SRTP layer, a somewhat complex
transform is required to allow only the selected header extension values to be encrypted. One of the most popular SRTP implementations had a significant bug in this area that was not detected for five years.

Second, it only protects the header extension values, and not their ids or lengths. It also does not protect the CSRCs. As noted above, this leaves a fair amount of potentially sensitive information exposed.

Third, it bloats the header extension space. Because each extension must be offered in both unencrypted and encrypted forms, twice as many header extensions must be offered, which will in many cases push implementations past the 14-extension limit for the use of one-byte extension headers defined in [RFC8285]. Accordingly, implementations will need to use two-byte headers in many cases, which are not supported well by some existing implementations.

Finally, the header extension bloat combined with the need for backwards compatibility results in additional wire overhead. Because two-byte extension headers may not be handled well by existing implementations, one-byte extension identifiers will need to be used for the unencrypted (backwards compatible) forms, and two-byte for the encrypted forms. Thus, deployment of [RFC6904] encryption for header extensions will typically result in multiple extra bytes in each RTP packet, compared to the present situation.

1.3. Goals

From this analysis we can state the desired properties of a solution:

* Build on existing [RFC3711] SRTP framework (simple to understand)
* Build on existing [RFC8285] header extension framework (simple to implement)
* Protection of header extension ids, lengths, and values
* Protection of CSRCs when present
* Simple signaling
* Simple crypto transform and SRTP interactions
* Backward compatible with unencrypted endpoints, if desired
* Backward compatible with existing RTP tooling
The last point deserves further discussion. While we considered possible solutions that would have encrypted more of the RTP header (e.g., the number of CSRCs), we felt the inability to parse the resultant packets with current tools, as well as additional complexity incurred, outweighed the slight improvement in confidentiality.

2. Terminology

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 [BCP14] RFC2119 RFC8174 when, and only when, they appear in all capitals, as shown here.

3. Design

This specification proposes a mechanism to negotiate encryption of all RTP header extensions (ids, lengths, and values) as well as CSRC values. It reuses the existing SRTP framework, is accordingly simple to implement, and is backward compatible with existing RTP packet parsing code, even when support for the mechanism has been negotiated.

4. Signaling

In order to determine whether the mechanism defined in this specification is supported, this document defines a new "a=cryptex" Session Description Protocol (SDP) [RFC4566] attribute to indicate support.

This attribute takes no value, and can be used at the session level or media level.

The presence of this attribute in the SDP (either in an offer or answer) indicates that the endpoint is capable of receiving RTP packets encrypted with Cryptex, as defined below.

Once each peer has verified that the other party supports receiving RTP packets encrypted with Cryptex, senders can unilaterally decide whether to use the Cryptex mechanism or not.

If BUNDLE is in use and the a=cryptex attribute is present for a media line, it MUST be present for all media lines belonging to the same bundle group. This ensures that the encrypted MID header extensions used to demux BUNDLE can be processed correctly. When used with BUNDLE, this attribute is assigned to the TRANSPORT category [RFC8859].
It is possible to signal and negotiate both Encryption of Header Extensions as defined in [RFC6904] and cryptex in the SDP O/A, however if a packet is encrypted with cryptex, it MUST NOT use the [RFC6904] header extension encryption mechanisms.

5. RTP Header Processing

[RFC8285] defines two values for the "defined by profile" field for carrying one-byte and two-byte header extensions. In order to allow a receiver to determine if an incoming RTP packet is using the encryption scheme in this specification, two new values are defined:

* 0xC0DE for the encrypted version of the one-byte header extensions (instead of 0xBEDE).
* 0xC2DE for the encrypted versions of the two-byte header extensions (instead of 0x100).

In the case of using two-byte header extensions, the extension id with value 256 MUST NOT be negotiated, as the value of this id is meant to be contained in the "appbits" of the "defined by profile" field, which are not available when using the values above.

If the "a=extmap-allow-mixed" attribute defined in [RFC8285] is negotiated, either one-byte or two-byte header ids can be used (with the values above), as in [RFC8285].

5.1. Sending

When the mechanism defined by this specification has been negotiated, sending a RTP packet that has any CSRCs or contains any (RFC8285) header extensions follows the steps below. This mechanism MUST NOT be used with header extensions other than the [RFC8285] variety.

If the packet contains solely one-byte extension ids, the 16-bit RTP header extension tag MUST be set to 0xC0DE to indicate that the encryption has been applied, and the one-byte framing is being used. If the packet contains only two-byte extension ids, the header extension tag MUST be set to 0xC2DE to indicate encryption has been applied, and the two-byte framing is being used.

If the packet contains CSRCs but no header extensions, an empty extension block consisting of the 0xC0DE tag and a 16-bit length field set to zero (explicitly permitted by [RFC3550]) MUST be appended, and the X bit MUST be set to 1 to indicate an extension block is present. This is necessary to provide the receiver an indication that the CSRCs in the packet are encrypted.
The RTP packet MUST then be encrypted as described in Encryption Procedure.

5.2. Receiving

When receiving an RTP packet that contains header extensions, the "defined by profile" field MUST be checked to ensure the payload is formatted according to this specification. If the field does not match one of the values defined above, the implementation MUST instead handle it according to the specification that defines that value. The implementation MAY stop and report an error if it considers use of this specification mandatory for the RTP stream.

If the RTP packet passes this check, it is then decrypted according to Decryption Procedure, and passed to the next layer to process the packet and its extensions. In the event that a zero-length extension block was added as indicated above, it can be left as-is and will be processed normally.

6. Encryption and Decryption

6.1. Packet Structure

When this mechanism is active, the SRTP packet is protected as follows:
specifically, the encrypted portion must include any CSRC identifiers, any RTP header extension (except for the first 4 bytes), and the RTP payload.

6.2. Encryption Procedure

The encryption procedure is identical to that of [RFC3711] except for the region to encrypt, which is as shown in the section above.

To minimize changes to surrounding code, the encryption mechanism can choose to replace a "defined by profile" field from [RFC8285] with its counterpart defined in RTP Header Processing above and encrypt at the same time.

For AEAD ciphers (e.g., GCM), the 12-byte fixed header and the four-byte header extension header (the "defined by profile" field and the length) are considered AAD, even though they are non-contiguous in the packet if CSRCs are present.

Uberti, et al. Expires 8 September 2022 [Page 8]
6.3. Decryption Procedure

The decryption procedure is identical to that of [RFC3711] except for the region to decrypt, which is as shown in the section above.

To minimize changes to surrounding code, the decryption mechanism can choose to replace the "defined by profile" field with its no-encryption counterpart from [RFC8285] and decrypt at the same time.

7. Backwards Compatibility

This specification attempts to encrypt as much as possible without interfering with backwards compatibility for systems that expect a certain structure from an RTPv2 packet, including systems that perform demultiplexing based on packet headers. Accordingly, the first two bytes of the RTP packet are not encrypted.

This specification also attempts to reuse the key scheduling from SRTP, which depends on the RTP packet sequence number and SSRC identifier. Accordingly these values are also not encrypted.

8. Security Considerations

This specification extends SRTP by expanding the portion of the packet that is encrypted, as shown in Packet Structure. It does not change how SRTP authentication works in any way. Given that more of the packet is being encrypted than before, this is necessarily an improvement.

The RTP fields that are left unencrypted (see rationale above) are as follows:

* RTP version
* padding bit
* extension bit
* number of CSRCs
* marker bit
* payload type
* sequence number
* timestamp
* SSRC identifier
* number of [RFC8285] header extensions

These values contain a fixed set (i.e., one that won’t be changed by extensions) of information that, at present, is observed to have low sensitivity. In the event any of these values need to be encrypted, SRTP is likely the wrong protocol to use and a fully-encapsulating protocol such as DTLS is preferred (with its attendant per-packet overhead).

9. IANA Considerations

9.1. cryptex SDP Attribute

This document updates the "Session Description Protocol Parameters" registry as specified in Section 8.2.4 of [RFC8866]. Specifically, it adds the SDP ‘cryptex’ attribute to the table for SDP media-level attributes.

Contact name: TBD
Contact email address: TBD
Attribute name: cryptex
Attribute syntax: This attribute takes no values.
Attribute semantics: N/A
Attribute value: N/A
Usage level: media-level
Charset dependent: No

Purpose: The presence of this attribute in the SDP indicates that the endpoint is capable of receiving RTP packets encrypted with Cryptex as described in this document. O/A procedures: SDP O/A procedures are described in Section 4 of this document.

Mux Category: TRANSPORT
10. Acknowledgements

The authors wish to thank Lennart Grahl for pointing out many of the issues with the existing header encryption mechanism, as well as suggestions for this proposal. Thanks also to Jonathan Lennox, Inaki Castillo, and Bernard Aboba for their review and suggestions.

11. References

11.1. Normative References


<https://www.rfc-editor.org/info/bcp14>


11.2. Informative References


Appendix A. Test Vectors

All values are in hexadecimal and represented in network order (big endian).

A.1. AES-CTR

Common values are organized as follows:

Rollover Counter: 00000000
Master Key: elf97a0d3e018be0d64fa32c06de4139
Master Salt: 0ec675ad498afeebbb960b3aabe6
Crypto Suite: AES_CM_128_HMAC_SHA1_80
Session Key: c61e7a93744f39ee10734afe3ff7a087
Session Salt: 30cbbc08863d8c85d49db34a9ae1
Authentication Key: cebe321f6ff7716b6fd4ab49af256a156d38baa4

A.1.1. RTP Packet with 1-byte header extension

RTP Packet:
A.1.2. RTP Packet with 2-byte header extension

RTP Packet:

900f1236
decafbad
cafebabe
t0000001
05020002
abababab
abababab
abababab
abababab

Encrypted RTP Packet:
A.1.3. RTP Packet with 1-byte header extension and CSRC fields

RTP Packet:

```
920f1238
decafbad
cafebabe
c2de0001
4ed9cc4e
6a712b30
96c5ca77
339d4204
ce0d7739
6cab6958
5fbc381
94a5
```

Encrypted RTP Packet:

```
920f1238
decafbad
cafebabe
8bb6e12b
5cffe6dd
c0de0001
92838c8c
09e58393
e1de3a9a
74734d67
45671338
c3acf11d
a2df8423
bee0
```
A.1.4. RTP Packet with 2-byte header extension and CSRC fields

RTP Packet:

```
920f1239
decafbad
cafebabe
0001e240
0000b26e
10000001
05020002
abababab
abababab
abababab
```

Encrypted RTP Packet:

```
920f1239
decafbad
cafebabe
f70e513e
b90b9b25
c2de0001
bbed4848
f4a64466
5f3d7f34
125914e9
f4d0ae92
3c6f479b
95a0f7b5
3133
```

A.1.5. RTP Packet with empty 1-byte header extension and CSRC fields

RTP Packet:

```
920f123a
decafbad
cafebabe
0001e240
0000b26e
bede0000
abababab
abababab
abababab
abababab
```

```
Encrypted RTP Packet:

920f123a
decafbad
cafebabe
7130b6ab
fe2ab0e3
c0de0000
e3d9f64b
25c9e74c
b4cf8e43
fb92e378
1c2c0cea
b6b3a499
a14c

A.1.6. RTP Packet with empty 2-byte header extension and CSRC fields

RTP Packet:

920f123b
decafbad
cafebabe
0001e240
0000b26e
10000000
abababab
abababab
abababab
abababab

Encrypted RTP Packet:

920f123b
decafbad
cafebabe
cbf24c12
4330e1c8
c2de0000
599dd45b
c9d687b6
03e8b59d
771fd38e
88b170e0
cd31e125
eabe
A.2. AES-GCM

Common values are organized as follows:

<table>
<thead>
<tr>
<th>Value</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rollover Counter:</td>
<td>00000000</td>
</tr>
<tr>
<td>Master Key:</td>
<td>00102030405060708090a0b0c0d0e0f</td>
</tr>
<tr>
<td>Master Salt:</td>
<td>a0a1a2a3a4a5a6a7a8a9aaab</td>
</tr>
<tr>
<td>Crypto Suite:</td>
<td>AEAD_AES_128_GCM</td>
</tr>
<tr>
<td>Session Key:</td>
<td>077c6143cb21bc35ff23d5f984a16e</td>
</tr>
<tr>
<td>Session Salt:</td>
<td>9af3e95364ebac9c99c5a7c4</td>
</tr>
</tbody>
</table>

A.2.1. RTP Packet with 1-byte header extension

RTP Packet:

```
900f1235
decafbad
cafebabe
bede0001
51000200
abababab
abababab
abababab
```

Encrypted RTP Packet:

```
900f1235
decafbad
cafebabe
c0de0001
39972dc9
572c4d99
e8fc355d
e743fb2e
94f9d8ff
54e72f41
93bbc5c7
4ffab0fa
9fa0fbeb
```

A.2.2. RTP Packet with 2-byte header extension

RTP Packet:

...
900f1236
decafbad
cafebabe
10000001
05020002
abababab
abababab
abababab
abababab

Encrypted RTP Packet:

900f1236
decafbad
cafebabe
c2de0001
bb75a4c5
45cd1f41
3bde7d6a
2b1e3263
de313667
c9632490
81b35a65
f5cb6c88
b394235f

A.2.3. RTP Packet with 1-byte header extension and CSRC fields

RTP Packet:

920f1238
decafbad
cafebabe
0001e240
0000b26e
bede0001
51000200
abababab
abababab
abababab
abababab

Encrypted RTP Packet:
A.2.4. RTP Packet with 2-byte header extension and CSRC fields

RTP Packet:

```
920f1239
deafbad
cafebabe
0001e240
0000b26e
10000001
05020002
abababab
abababab
abababab
abababab
```

Encrypted RTP Packet:

```
```
A.2.5. RTP Packet with empty 1-byte header extension and CSRC fields

RTP Packet:

```
920f123a
decafbad
cafebabe
3680524f
8d312b00
c2de0001
c78d1200
38422bc1
11a7187a
18246f98
0c059cc6
bc9df8b6
26394eca
344e4b05
d80fea83
```

Encrypted RTP Packet:

```
920f123a
decafbad
cafebabe
15b6bb43
37906fff
c0de0000
b7b96453
7a2b03ab
7ba5389c
e9331712
6b5d974d
f30c6884
dcb651c5
e120c1da
```
A.2.6. RTP Packet with empty 2-byte header extension and CSRC fields

RTP Packet:

920f123b
decafbad
cafebabe
0001e240
0000b26e
10000000
abababab
abababab
abababab
abababab

Encrypted RTP Packet:

920f123b
decafbad
cafebabe
dcb38c9e
48bf95f4
c2de0000
61ee432c
f9203170
76613258
d3ce4236
c06ac429
681ad084
13512dc9
8b5207d8

Authors’ Addresses

Justin Uberti
Clubhouse
Email: justin@uberti.name

Cullen Jennings
Cisco
Email: fluffy@iii.ca

Sergio Garcia Murillo
CoSMo
Email: sergio.garcia.murillo@cosmosoftware.io
Multiplexing Scheme Updates for QUIC
draft-ietf-avcore-rfc7983bis-04.txt

Abstract

This document defines how QUIC, Datagram Transport Layer Security (DTLS), Real-time Transport Protocol (RTP), RTP Control Protocol (RTCP), Session Traversal Utilities for NAT (STUN), Traversal Using Relays around NAT (TURN), and ZRTP packets are multiplexed on a single receiving socket.

This document updates RFC 7983 and RFC 5764.

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 http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on November 13, 2022.
Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the
document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal
Provisions Relating to IETF Documents
(http://trustee.ietf.org/license-info) in effect on the date of
publication of this document. Please review these documents
carefully, as they describe your rights and restrictions with respect
to this document. Code Components extracted from this document must
include Simplified BSD License text as described in Section 4.e of
the Trust Legal Provisions and are provided without warranty as
described in the Simplified BSD License.

Table of Contents

1. Introduction ........................................... 3
   1.1. Terminology ...................................... 3
2. Multiplexing of TURN Channels .......................... 3
3. Updates to RFC 7983 .................................. 4
4. Security Considerations ................................. 5
5. IANA Considerations .................................. 6
6. References ............................................ 6
   6.1. Normative References ............................. 6
   6.2. Informative References ........................... 7
Acknowledgements .......................................... 8
Authors’ Addresses ......................................... 8
1. Introduction


The multiplexing scheme described in this document enables multiple usage scenarios. Peer-to-peer QUIC in WebRTC scenarios, described in [P2P-QUIC] [P2P-QUIC-TRIAL], uses RTP for transport of audio and video along with QUIC for data exchange. For this use case, SRTP [RFC3711] is keyed using DTLS-SRTP [RFC5764] and therefore SRTP/SRTCP [RFC3550], STUN, TURN, DTLS and QUIC need to be multiplexed on the same port. Were SRTP to be keyed using QUIC-SRTP, SRTP/SRTCP, STUN, TURN and QUIC would need to be multiplexed on the same port. Where QUIC is used for peer-to-peer transport of data as well as RTP [I-D.engelbart-rtp-over-quic] STUN, TURN and QUIC need to be multiplexed on the same port.

The scheme described in this document is compatible with QUIC version 2 [I-D.ietf-quic-v2]. However, it is not compatible with QUIC Bit greasing, as defined in [I-D.ietf-quic-bit-grease]. Therefore, in situations where multiplexing is desired, QUIC Bit greasing MUST NOT be negotiated.

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Multiplexing of TURN Channels

TURN channels are an optimization where data packets are exchanged with a 4-byte prefix instead of the standard 36-byte STUN overhead (see Section 3.5 of [RFC8656]). [RFC7983] allocated the values from 64 to 79 in order to allow TURN channels to be demultiplexed when the TURN Client does the channel binding request in combination with the demultiplexing scheme described in [RFC7983].

When QUIC Bit greasing is not negotiated, the first octet of a QUIC short header packet falls in the range 64 to 127, thereby overlapping with the allocated range for TURN channels of 64 to 79.
Where QUIC Bit greating is not negotiated, the first octet of QUIC long header packets fall in the range 192 to 255. Since QUIC long header packets preceed QUIC short header packets, if no packets with a first octet in the range of 192 to 255 have been received, a packet whose first octet is in the range of 64 to 79 can be demultiplexed unambiguously as TURN Channel traffic. Since WebRTC implementations supporting QUIC data exchange do not utilize TURN Channels, once packets with a first octet in the range of 192 to 255 have been received, a packet whose first octet is in the range of 64 to 127 can be demultiplexed as QUIC traffic.

3. Updates to RFC 7983

This document updates the text in Section 7 of [RFC7983] (which in turn updates [RFC5764]) as follows:

OLD TEXT

The process for demultiplexing a packet is as follows. The receiver looks at the first byte of the packet. If the value of this byte is in between 0 and 3 (inclusive), then the packet is STUN. If the value is between 16 and 19 (inclusive), then the packet is ZRTP. If the value is between 20 and 63 (inclusive), then the packet is DTLS. If the value is between 64 and 79 (inclusive), then the packet is TURN Channel. If the value is in between 128 and 191 (inclusive), then the packet is RTP (or RTCP, if both RTCP and RTP are being multiplexed over the same destination port). If the value does not match any known range, then the packet MUST be dropped and an alert MAY be logged. This process is summarized in Figure 3.

---Diagram---

```plaintext
+----------------+
|        [0..3] -+--> forward to STUN
|                |
|      [16..19] -+--> forward to ZRTP
|                |
|      [20..63] -+--> forward to DTLS
|                |
|      [64..79] -+--> forward to TURN Channel
|                |
|    [128..191] -+--> forward to RTP/RTCP
+----------------+
```

Figure 3: The DTLS-SRTP receiver’s packet demultiplexing algorithm.

END OLD TEXT
NEW TEXT

The process for demultiplexing a packet is as follows. The receiver looks at the first byte of the packet. If the value of this byte is in between 0 and 3 (inclusive), then the packet is STUN. If the value is between 16 and 19 (inclusive), then the packet is ZRTP. If the value is between 20 and 63 (inclusive), then the packet is DTLS. If the value is in between 128 and 191 (inclusive) then the packet is RTP (or RTCP, if both RTCP and RTP are being multiplexed over the same destination port). If the value is between 80 and 127 or between 192 and 255 (inclusive) then the packet is QUIC. If the value is between 64 and 79 inclusive, then if a packet has been previously forwarded that is in the range of 192 and 255, then the packet is QUIC, otherwise it is TURN Channel.

If the value does not match any known range, then the packet MUST be dropped and an alert MAY be logged. This process is summarized in Figure 3.

```
+--------------------+
| [0..3] ----> forward to STUN |
| [16..19] ----> forward to ZRTP |
| [20..63] ----> forward to DTLS |
| [64..79] ----> forward to TURN Channel |
| [64..127] ----> forward to QUIC  |
| (Short Header) |
| [128..191] ----> forward to RTP/RTCP |
| [192..255] ----> forward to QUIC  |
| (Long Header) |
```

Figure 3: The receiver’s packet demultiplexing algorithm.

Note: The demultiplexing of QUIC packets requires that QUIC Bit greasing [I-D.ietf-quic-bit-grease] not be negotiated.

END NEW TEXT

4. Security Considerations

The solution discussed in this document could potentially introduce some additional security considerations beyond those detailed in [RFC7983]. Due to the additional logic required, if mis-implemented, heuristics have the potential to mis-classify packets.
When QUIC is used only for data exchange, the TLS-within-QUIC exchange [RFC9001] derives keys used solely to protect the QUIC data packets. If properly implemented, this should not affect the transport of SRTP nor the derivation of SRTP keys via DTLS-SRTP. However, were the TLS-within-QUIC exchange to be used to derive SRTP keys, both transport and SRTP key derivation could be adversely impacted by a vulnerability in the QUIC implementation.

5. IANA Considerations

This document does not require actions by IANA.

6. References

6.1. Normative References

[I-D.ietf-quic-bit-grease]


6.2. Informative References

[I-D.engelbart-rtp-over-quic]
Ott, J. and M. Engelbart, "RTP over QUIC", draft-engelbart-rtp-over-quic-02 (work in progress), March 7, 2022.

[I-D.ietf-quic-v2]

[RFC6189]

[P2P-QUIC]

[P2P-QUIC-TRIAL]
Hampson, S., "RTCQuicTransport Coming to an Origin Trial Near You (Chrome 73)", January 2019,
Acknowledgments

We would like to thank Martin Thomson, Roni Even and other participants in the IETF QUIC and AVTCORE working groups for their discussion of the QUIC multiplexing issue, and their input relating to potential solutions.

Authors’ Addresses

Bernard Aboba
Microsoft Corporation
One Microsoft Way
Redmond, WA  98052
USA

Email: bernard.aboba@gmail.com

Gonzalo Salgueiro
Cisco Systems
7200-12 Kit Creek Road
Research Triangle Park, NC  27709
United States of America

Email: gsalguei@cisco.com

Colin Perkins
School of Computing Science
University of Glasgow
Glasgow  G12 8QQ
United Kingdom

Email: csp@csperkins.org
RTP Payload Format for the SCIP Codec
draft-ietf-avtcore-rtp-scip-00.txt

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
http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on August 22, 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.
Abstract

This document describes the RTP payload format of the Secure Communication Interoperability Protocol (SCIP) as audio and video media subtypes. It provides RFC 6838 compliant media subtype definitions. SCIP-214.2 and SCIP-210 describe the protocols that comprise the SCIP RTP packet payload. This document follows the registration for related media types called "audio/scip" and "video/scip" with IANA and formatted according to RFC 4855.

Table of Contents

1. Introduction ..............................................2
   1.1. Conventions ...........................................3
   1.2. Abbreviations ........................................3
2. Background ................................................3
3. Media Format Description ..................................4
4. Payload Format ............................................5
   4.1. RTP Header Fields ....................................5
5. Payload Format Parameters .................................5
   5.1. Media Subtype "audio/scip" ............................6
   5.2. Media Subtype "video/scip" ...........................7
   5.3. Mapping to SDP ........................................8
   5.4. SDP Offer/Answer Considerations ......................9
6. Security Considerations ...................................9
7. IANA Considerations ......................................9
8. References ................................................9
   8.1. Normative References ................................9
   8.2. Informative References ..............................11
9. Authors’ Addresses .......................................12

1. Introduction

The IANA registration of media subtype types in the IETF tree created two similar media subtypes "scip" under the audio and video media types [AUDIOSCIP], [VIDEOSCIP]. This document, as the common top-level reference, provides information on their similarities and differences and the usage of those media subtypes.

This document details usage of the scip pseudo-codec as a secure session establishment protocol and transport protocol over RTP. It provides a reference for network security policymakers, network equipment OEMs, procurement personnel, and government agency and commercial industry representatives.
1.1. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

Best current practices for writing an RTP payload format specification were followed [RFC2736] [RFC8088].

1.2. Abbreviations

The following abbreviations are used in this document.

- AVP: Audio/Video Profile
- DTX: Discontinuous Transmission
- FNBDT: Future Narrowband Digital Terminal
- ICWG: Interoperability Control Working Group
- IICWG: International Interoperability Control Working Group
- NATO: North Atlantic Treaty Organization
- SCIP: Secure Communication Interoperability Protocol
- SDP: Session Description Protocol

2. Background

The Secure Communication Interoperability Protocol (SCIP) allows the negotiation of several voice, data, and video applications using various encryption suites. SCIP also provides several important characteristics that have led to its broad acceptance in the international user community. These features include end-to-end security at the application layer, authentication of user identity, the ability to apply different security levels for each secure session, and secure communication over any end-to-end data connection.

SCIP began in the U.S. as the Future Narrowband Digital Terminal (FNBDT) Protocol. A combined Department of Defense and vendor consortium formed a governing organization named the Interoperability Control Working Group (ICWG) to manage the protocol. In time, the group expanded to include NATO, NATO partners and European vendors under the name International Interoperability Control Working Group (IICWG), which was later renamed the SCIP Working Group.
SCIP is presently implemented in U.S. and NATO secure voice, video, and data products operating on commercial, private, and tactical IP networks worldwide using the scip media subtype. First generation SCIP devices operated on circuit-switched networks. SCIP was then expanded to radio and IP networks. The scip media subtype transports SCIP secure session establishment signaling and secure application traffic. The built-in negotiation and flexibility provided by the SCIP standards make it a natural choice for many scenarios that require various secure applications and associated encryption suites. SCIP has been endorsed by many nations as the secure end-to-end solution for secure voice, video, and data devices. SCIP standards are currently available to participating government/military communities and select OEMs of equipment that support SCIP.

However, SCIP must operate over global networks (including private and commercial networks). Without access to necessary information to support SCIP, some networks may not support the SCIP media subtypes. Issues may occur simply because information is not as readily available to OEMs, network administrators, and network architects.

This RFC provides essential information about audio/scip and video/scip media subtypes that enables network equipment manufacturers to include scip as a known audio and video media subtype in their equipment and enables network administrators to define and implement a compatible security policy.

All current IP-based SCIP devices support "scip" as a media subtype. Registration of scip as a media subtype provides a common reference for network equipment manufacturers to recognize SCIP in a payload declaration.

3. Media Format Description

The "scip" media subtype indicates support for and identifies SCIP traffic that is being transferred using RTP. SCIP traffic requires end-to-end bit integrity, therefore transcoding SHALL NOT be performed over the end-to-end IP connection. The audio/scip and video/scip media subtype data streams within the network, including the VoIP network, MUST be a transparent relay and be treated as "clear-channel data", similar to the Clearmode media subtype defined by RFC 4040. However, Clearmode is defined as a gateway protocol and limited to a sample rate of 8000 Hz and 64kbps bandwidth only [RFC4040].
Clearmode is not defined for the higher sample and data rates required for some SCIP traffic.

4. Payload Format

The RTP Packet content of SCIP traffic is dependent upon the SCIP session state. SCIP secure session establishment uses protocols defined in SCIP-210 [SCIP210] to negotiate an application. SCIP secure traffic may consist of the encrypted output of codecs such as MELPe [RFC8130], G.729D [RFC3551], H.264 [RFC6184], or other media encodings, based on the application negotiated during SCIP secure session establishment. SCIP traffic is highly variable and may include other SCIP signaling information in the media stream. SCIP traffic may not always be a continuous stream at the bit rate specified in the SDP [RFC8866] since discontinuous transmission (DTX) or other mechanisms may be used. The SCIP payload size will vary, especially during SCIP secure session establishment.

4.1. RTP Header Fields

The SCIP RTP header fields SHALL conform to RFC 3550.

SCIP traffic may be continuous or discontinuous. The Timestamp field increments based on the sampling clock for discontinuous transmission as described in [RFC3550], Section 5.1. The Timestamp field for continuous transmission applications is dependent on the sampling rate of the media as specified in the media subtype’s specification (e.g., MELPe [RFC8130]). Note that during a call, both discontinuous and continuous traffic are highly probable. Therefore, a jitter buffer MAY be implemented in endpoint devices only but SHOULD NOT be implemented in network devices. Additionally, network devices SHOULD NOT repacketize SCIP packets.

The Marker bit SHALL be set to zero for discontinuous traffic. The Marker bit for continuous traffic is based on the underlying media subtype specification. The underlying media is opaque within SCIP RTP packets.

5. Payload Format Parameters

The SCIP RTP payload format is identified using the scip media subtype, which is registered in accordance with [RFC4855] and per the media type registration template form [RFC6838]. A clock rate of 8000 Hz SHALL be used for "audio/scip". A clock rate of 90000 Hz SHALL be used for "video/scip".
5.1. Media Subtype "audio/scip"

Media type name: audio

Media subtype name: SCIP

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: Binary. This media subtype is only defined for transfer via RTP. There SHALL be no encoding/decoding (transcoding) of the audio stream as it traverses the network.


Interoperability considerations: N/A

Published specifications: [SCIP214], [SCIP210]

Applications which use this media: N/A

Fragment Identifier considerations: none

Restrictions on usage: N/A

Additional information:

1. Deprecated alias names for this type: N/A
2. Magic number(s): N/A
3. File extension(s): N/A
4. Macintosh file type code: N/A
5. Object Identifiers: N/A

Person to contact for further information:

1. Name: Michael Faller and Daniel Hanson
2. Email: michael.faller@gd-ms.com and dan.hanson@gd-ms.com

Intended usage: Common, Government and Military
5.2. Media Subtype "video/scip"

Media type name: video
Media subtype name: scip
Required parameters: N/A
Optional parameters: N/A

Encoding considerations: Binary. This media subtype is only defined for transfer via RTP. There SHALL be no encoding/decoding (transcoding) of the video stream as it traverses the network.


Interoperability considerations: N/A

Published specifications: [SCIP214], [SCIP210]

Applications which use this media: N/A

Fragment Identifier considerations: none

Restrictions on usage: N/A

Additional information:

1. Deprecated alias names for this type: N/A
2. Magic number(s): N/A
3. File extension(s): N/A
4. Macintosh file type code: N/A
5. Object Identifiers: N/A

Person to contact for further information:

1. Name: Michael Faller and Daniel Hanson
2. Email: michael.faller@gd-ms.com and dan.hanson@gd-ms.com

Intended usage: Common, Government and Military

Authors:

Michael Faller - michael.faller@gd-ms.com

Daniel Hanson - dan.hanson@gd-ms.com

Change controller:

SCIP Working Group - ncia.cis3@ncia.nato.int

5.3. Mapping to SDP

The mapping of the above defined payload format media subtype and its parameters SHALL be done according to Section 3 of [RFC4855].

An example mapping for audio/scip is:

```
m=audio 50000 RTP/AVP 96  
a=rtpmap:96 scip/8000
```

An example mapping for video/scip is:

```
m=video 50002 RTP/AVP 97  
a=rtpmap:97 scip/90000
```

An example mapping for both audio/scip and video/scip is:

```
m=audio 50000 RTP/AVP 96  
a=rtpmap:96 scip/8000
m=video 50002 RTP/AVP 97  
a=rtpmap:97 scip/90000
```

The application negotiation between endpoints will determine whether the audio and video streams are transported as separate
streams over the audio and video payload types or as a single media stream on the video payload type.

5.4. SDP Offer/Answer Considerations

In accordance with the SDP Offer/Answer model [RFC3264], the SCIP device SHALL list the SCIP payload type in order of preference in the "m" media line.

6. Security Considerations

RTP packets using the payload format defined in this specification are subject to the security considerations discussed in the RTP specification [RFC3550], and in any applicable RTP profile such as RTP/AVP [RFC3551], RTP/AVPF [RFC4585], RTP/SAVPF [RFC3711], or RTP/SAVP [RFC5124]. However, as "Securing the RTP Protocol Framework: Why RTP Does Not Mandate a Single Media Security Solution" [RFC7202] discusses, it is not an RTP payload format’s responsibility to discuss or mandate what solutions are used to meet the basic security goals like confidentiality, integrity, and source authenticity for RTP in general. This responsibility lays on anyone using RTP in an application. They can find guidance on available security mechanisms and important considerations in "Options for Securing RTP Sessions" [RFC7201]. Applications SHOULD use one or more appropriate strong security mechanisms. The rest of this Security Considerations section discusses the security impacting properties of the payload format itself.

This RTP payload format and its media decoder do not exhibit any significant non-uniformity in the receiver-side computational complexity for packet processing, and thus are unlikely to pose a denial-of-service threat due to the receipt of pathological data. Nor does the RTP payload format contain any active content.

7. IANA Considerations

The audio/scip and video/scip media subtypes have been registered with IANA [AUDIOSCIP] [VIDEOSCIP].

8. References

8.1. Normative References

[AUDIOSCIP] Faller, M., and D. Hanson, "audio/scip", Internet Assigned Numbers Authority (IANA), 28 January 2021,
<https://www.iana.org/assignments/media-types/audio/scip>.


8.2. Informative References


9. Authors’ Addresses

Daniel Hanson
General Dynamics Mission Systems, Inc.
150 Rustcraft Road
Dedham, MA 02026, USA
E-mail: dan.hanson@gd-ms.com

Michael Faller
General Dynamics Mission Systems, Inc.
150 Rustcraft Road
Dedham, MA 02026, USA
E-mail: michael.faller@gd-ms.com

Keith Maver
General Dynamics Mission Systems, Inc.
150 Rustcraft Road
Dedham, MA 02026, USA
E-mail: keith.maver@gd-ms.com

SCIP Working Group, CIS3 Partnership
NATO Communications and Information Agency
Oude Waalsdorperweg 61, 2597AK
The Hague, The Netherlands
E-mail: ncia.cis3@ncia.nato.int
Game State over Real Time Protocol
draft-jennings-dispatch-game-state-over-rtp-00

Abstract

This specification defines an Real Time Protocol (RTP) payload to
send game moves and the state of game objects over RTP. This is
useful for games as well collaboration systems that use augment or
virtual reality.

RTP provide a way to synchronize game state between players with
robust technique for recovery from network packet loss while still
having low latency.

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/.

Internet-Drafts are draft documents valid for a maximum of six months
and may be updated, replaced, or obsoleted by other documents at any
time. It is inappropriate to use Internet-Drafts as reference
material or to cite them other than as "work in progress."

This Internet-Draft will expire on 5 September 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the
document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal
license-info) in effect on the date of publication of this document.
Please review these documents carefully, as they describe your rights
and restrictions with respect to this document.
Table of Contents

1. Overview ........................................... 3
2. Goals: ........................................... 3
3. Primitives ........................................... 4
   3.1. Location ........................................... 4
   3.2. Scale ........................................... 4
   3.3. Normal ........................................... 5
   3.4. TextureUV ........................................... 5
   3.5. Rotation ........................................... 5
   3.6. Child Transform ................................... 6
   3.7. Texture URL ......................................... 6
   3.8. Mesh URL ........................................... 6
   3.9. Texture Stream ..................................... 6
   3.10. Timestamp ......................................... 6
4. Objects ............................................. 7
   4.1. Common Objects ..................................... 7
      4.1.1. Generic Game Object ................................ 7
      4.1.2. Player Head ....................................... 7
      4.1.3. Mesh ........................................... 8
      4.1.4. External Mesh ..................................... 8
      4.1.5. Player Hand ...................................... 9
5. Encoding ............................................ 10
   5.1. Tag .............................................. 10
   5.2. Float ............................................ 11
   5.3. Boolean .......................................... 11
   5.4. Integer .......................................... 11
   5.5. String .......................................... 12
   5.6. Blob ............................................ 12
6. Full Intra Request .................................. 12
7. IANA ............................................... 12
8. RTP ................................................ 12
   8.1. Game State Tag Registry ............................. 12
9. Security ............................................. 13
   Appendix A. Acknowledgments ............................ 13
   Appendix B. Implementations ............................... 14
   Appendix C. Test Vectors ................................ 14
   C.1. Head Location ..................................... 14
   Appendix D. Encode API ................................... 15
   Appendix E. Decode API ................................... 15
   Appendix F. EBNF ....................................... 16
   Authors’ Addresses ...................................... 19
1. Overview

Many real time applications, such as games, want to share state about 3D objects across the network. This specification allows an application to define objects with state, and the current values of that state over RTP.

The conceptual model is each RTP sender has a small number of objects with state that needs to be synchronized to the other side. The current values are periodically sent over RTP. They MAY be sent when the values change, but they MUST also be sent periodically so that any lost updates are eventually received and state is consistent between the sender and receiver.

The state sent can include a time stamp and rate change estimates that allow the receiver to estimate the current state values even at a point in the future. An application that receives a state update can apply it immediately (often called immediate based), wait a fixed delay and then apply state changes (often called delay based), or apply a predicted value based on overwriting any previous predictions (often called rollback based).

In many cases the state does not have any units but if does, SI units SHOULD be used. Unless otherwise defined by the application, the default coordinate system SHOULD be a left handed coordinate system where Y points up and X points to the right.

Applications can define their own objects or use some of the predefined common objects. Each object is identified by a type and identifier number to uniquely identify the object within the scope of that sender’s RTP stream. Multiple updates and objects can be combined in a single RTP packet so that they are guaranteed to be fate shared and either atomically delivered at the same time or not delivered at all to the receiver.

The objects are defined as a series of primitives that define common types. The objects and updates to state are encoded with Tag Length Value (TLV) style encoding so that receivers can skip objects they do not understand. The Objects in an single RTP packet MUST be processed in order. This allows a sender to write state in an old format followed by a new format, allowing the new format to override values in the old format. This allows for easy upgrade of the protocol with backwards compatibility.

2. Goals:

* Support 2D and 3D
* Support delay and rollback based synchronization
* Relatively compact, simple encoding
* Extensible for applications to send custom data
* Support for forward and backwards compatibility

3. Primitives

This section defines primitives that are useful in defining objects. The definition are in W3C style EBNF [https://www.w3.org/TR/2010/REC-xquery-20101214/#EBNFNotation](https://www.w3.org/TR/2010/REC-xquery-20101214/#EBNFNotation).

3.1. Location

Loc1 ::= Float32 /* x */
   Float32 /* y */
   Float32 /* z */

Loc1 is simply a 3D location of a point stored in Float32;

Loc2 ::= Float32 /* x */
   Float32 /* y */
   Float32 /* z */
   Float16 /* vx */
   Float16 /* vy */
   Float16 /* vz */

Loc2 has a location as Float32 followed by the rate of change in location per second as Float16.

3.2. Scale

Scale1 ::= Float16 /* all dimensions */

Scale1 will scale the object in all dimensions equally.

Scale2 ::= Float32 /* x */
   Float32 /* y */
   Float32 /* z */
   Float16 /* vx */
   Float16 /* vy */
   Float16 /* vz */

Scale2 has a scale in each axis as Float32 followed by the rate of change in the scale per second as Float16.
3.3. Normal

Norm1 ::=  
Float16 /* nx */  
Float16 /* ny */  
Float16 /* nz */

Normal vector for a point.

3.4. TextureUV

TextureUV1 ::=  
VarUInt /* u */  
VarUInt /* v */

Location in texture map for a point.

3.5. Rotation

Rot1 ::=  
Float16 /* i */  
Float16 /* j */  
Float16 /* k */

The non-real parts of a normalized rotation quaternion. The real part can be computed based on how it is normalized.

Rot2 ::=  
Float16 /* s.i */  
Float16 /* s.j */  
Float16 /* s.k */  
Float16 /* e.i */  
Float16 /* e.j */  
Float16 /* e.k */

Rot2 defines the s, the current rotation, and e, an estimated value of the rotation in one second. The rotation e is chosen such that rotation of the object will follow a rotation along the great circle path from s to e with an constant angular rotation rate such that it would reach e in 1 second.

This representation of rate of change of the rotation allows the receiver to use an algorithm such as SLERP [https://en.wikipedia.org/wiki/Slerp (https://en.wikipedia.org/wiki/Slerp)] to estimate the current rotation for the object for short periods of time into the future. The representation cannot represent something that is rotating faster than one revolution ever 2 seconds.
Open Issue: Would there be a better way to represent angular velocity?

3.6. Child Transform

Transform1 ::= Float16 /* tx */
Float16 /* ty */
Float16 /* tz */

Defines a linear translation of a child object from a base object.

3.7. Texture URL

TextureUrl1 ::= String

URL of image with texture map. JPEG images SHOULD be supported.

3.8. Mesh URL

MeshUrl1 ::= String

URL of external mesh.

Open Issue: Any mandatory to implements mesh formats?

3.9. Texture Stream

In some cases it is desirable to provide a separate RTP video stream which might have the texture used be the frame from the video stream with the corresponding time to the game object that uses the texture map. To do this there needs to be a way to identify the other RTP video stream. One way to do that is use the "Payload Type" value used for the RTP packets for that video stream.

TextureRtpPT1 ::= UInt8 /* pt */

RTP "Payload Type" value of RTP video stream to use as a texture map.

Open Issue: Is there a better way to identity video stream?

3.10. Timestamp

Time1 ::= UInt16 /* time in ms */
Lower 16 bits of number in milliseconds since 00:00:00 UTC on 1
January 1970, not counting leap seconds. The assumption is these
timestamps are accurate to about the level of an NTP synchronized
clock. In C++20 this can be found with:

duration_cast<milliseconds>(
    system_clock::now().time_since_epoch()
).count() % 65536

4. Objects

All objects must start with a unique tag that defines the object
type, a VarUInt length, a VarUInt objectID, and the data for object.
Applications can reserve tags for objects in the registry defined in
the IANA section. Objects can be made extensible by adding a section
that contains optional tag, length, value tuples.

4.1. Common Objects

4.1.1. Generic Game Object

It is common to need to describe the location, rotation, scale, and
parent for objects in a scene.

Object1 ::= tagObject1 Length ObjectID Time1
Loc1
Rot1
Scale1
( tagParent1 Length ObjectID )? /* Optional Parent */

Object1 contains a simple 3D location, rotation, scale and an
optional ID of a parent object.

Object2 ::= tagObject2 Length ObjectID Timel
Loc2
Rot2
Scale2
( tagParent1 Length ObjectID )? /* Optional Parent */

Object2 has the same information but with the ability to scale
differently in each dimension and derivatives that describe how all
the parameters change over time.

4.1.2. Player Head

Head1 ::= tagHead1 Length ObjectID Timel
Loc2 Rot2
( tagHeadIpD Length Float16 /* IPD */ )?
Defines location and rotation of head with optional interpupillary distance (IPD).

4.1.3. Mesh

Object with the following variable length arrays:

Mesh1 ::= tagMesh1 Length ObjectID
  ( TextureUrl1 | TextureRtpPt1 )
  VarUInt /* num Vertexes */
  Loc1+ /* vertexes */
  VarUInt /* numNormals */
  Norm1* /* normals */
  VarUInt /* numTextureCoord */
  TextureUV1* /* textureCoord */
  VarUInt /* numTrianglesIndex */
  VarUInt+ /* trianglesIndex */

The vertex is an array of at least 3 locations that defines the vertex of a triangle mesh. The normals array can either be empty or the same size as the vertex and defines the normal for each vertex. The uv array must be empty or the same size as vertex array and have the u,v coordinate in the texture map for the vertex.

The texture can be defined by a URL that may refer to some local resource or a resource retrieved over the network. Alternatively, the texture can reference a local RTP video stream, in which case the most recently received frame of video is used as the texture and texture updates with new frames of video.

The triangles array can be of a different size from the vertex array. Each entry defines one triangle in the mesh and contains the index of the three vertexes in the vertexes array. Vertexes MUST be in counter clockwise order.

An important limitation to note is that objects cannot span RTP packets so the Mesh needs to be small enough that it size is less that the MTU. A typical limit might be as low as 50 triangles.

4.1.4. External Mesh

The Mesh2 object allows a mesh to be loaded from an external URL and then moved, rotated, and scaled. An optional texture map may be used.
Mesh2 := tagMesh2 Length ObjectID
   Loc2 Rot2 Scale2
   MeshUrl1
   ( TextureUrl1 | TextureRtpPT1 )? /* Optional Texture Map */
   ( tagParent1 Length ObjectID )? /* Optional Parent */

4.1.5. Player Hand

Hand1 := tagHand1 Length ObjectID Time1
   Boolean /* left */
   Loc2 Rot2

   The Hand1 identifies a location and rotation of a hand. The left is
   true for the left hand, false for a right hand.

Hand2 := tagHand2 Length ObjectID Time1
   Boolean /* left */
   Loc2 Rot2
   Transform1 /* wrist */
   Transform1 /* thumbTip */
   Transform1 /* thumbIP */
   Transform1 /* thumbMCP */
   Transform1 /* thumbCMC */
   Transform1 /* indexTip */
   Transform1 /* indexDIP */
   Transform1 /* indexPIP */
   Transform1 /* indexMCP */
   Transform1 /* indexCMC */
   Transform1 /* middleTip */
   Transform1 /* middleDIP */
   Transform1 /* middlePIP */
   Transform1 /* middleMCP */
   Transform1 /* middleCMC */
   Transform1 /* ringTip */
   Transform1 /* ringDIP */
   Transform1 /* ringPIP */
   Transform1 /* ringMCP */
   Transform1 /* ringCMC */
   Transform1 /* pinkyTip */
   Transform1 /* pinkyDIP */
   Transform1 /* pinkyPIP */
   Transform1 /* pinkyMCP */
   Transform1 /* pinkyCMC */

   Hand2 represents a wired skeletal hand. The boolean is true for the
   left hand. The location should represent the location of the palm
   and rotation is from the palm facing the x axis.
The transform points represent the relative location to the main joints in the fingers from the hand location. The location of the wrist is followed by finger joints. The fingers are ordered by thumb, index, middle, ring, then pinky. The joints are ordered by tip of finger (TIP), distal interphalangeal joint (DIP), proximal interphalangeal joint (PIP), metacarpophalangeal joint (MCP), then carpometacarpal joint (CMC). Note the thumb has no middle phalange so the PIP and DIP joint just become the interphalangeal joint (IP).

Note: The Microsoft documentation calls the MCP "knuckle" for the fingers and PIP "middle joint" and the CMC is called "Metacarpal", which is very confusing since this is not the metacarpophalangeal joint. The MCP for the thumb gets called "proximal", not knuckle, and the IP is "middle".

Names of the joints are explained in [https://en.wikipedia.org/wiki/Interphalangeal_joints_of_the_hand](https://en.wikipedia.org/wiki/Interphalangeal_joints_of_the_hand)

This is about 175 bytes, so at a 5Hz update rate this will be around 10 Kbps. [TODO: Check.]

5. Encoding

Each RTP payload will contain one or more objects. An object cannot be split across two RTP packets. The general design is that if the decoder has not been coded to understand a given object type, the decode can skip over the object to the next object but will not be able to provide any information and the internal format of the data.

The objects are defined such that they always start with a tag that indicates the type followed by a length of the object (so it can be skipped). Any optional or variable parts of the object also use tags so that the decoder can always be implemented as a LL(1) parser.

In general, network byte order encoding is used on the wire.

A length field is encoded to represent the number of bytes following the length field and does not include the size of the length field or information before it.

5.1. Tag

Constant values of tags can be found in the IANA section. They are encoded as VarUInt.
5.2. Float

Float16, Float32, and Float64 are encoded as IEEE 754 half, single, and double precisions respectively.

The half precision are often useful for things where only a few significant digits are needed such as normals. The internal representation of them will often be single precession (four bytes) in memory but they can be reduced to two bytes when encoded on the wire.

Note there is an example decode for a single precision float at

5.3. Boolean

Encoded as byte with 0 for false or 1 for true.

5.4. Integer

UInt8, Int8, UInt16, Int16, UInt32, Int32, UInt64, Int64 encoded as 1, 2, 4, or 8 bytes.

VarInt is encoded as:
* Top bits of first byte is 0, then 7 bit signed integer (-64 to 63)
* Top bits of first byte is 10, then 6+8 bit signed integer (-8192 to 8191)
* Top bits of first byte is 110, then 5+16 bit signed integer (1,048,576 to 1,048,575)
* Top bits of first byte is 1110,0001 then next 4 bytes 32 bit signed integer
* Top bits of first byte is 1110,0010 then next 8 bytes 64 bit signed integer

VarUInt is encoded as:
* Top bits of first byte is 0, then 7 bit unsigned integer
* Top bits of first byte is 10, then 6+8 bit unsigned integer
* Top bits of first byte is 110, then 5+16 bit unsigned integer
5.5. String

Strings are encoded as a VarUInt length in bytes (not characters) followed by a UTF-8 representation of the string.

5.6. Blob

Blobs are encoded as a VarUInt length in bytes followed by the binary data that goes in the blob.

6. Full Intra Request

RTP supports a Full Intra Request (FIR) Feedback Control feedback messages. When an RTP sender receives a FIR, it SHOULD send a copy of all the relevant game state.

7. IANA

8. RTP

This section can be split out a separate payload draft we need some extra work.

The media type is application/gamestate. There are no optional or required parameters. The RTP marker bit is not used. The RTP clock MUST be 90 kHz.

Multiple Objects as defined in this specification can be concatenated into one RTP payload.

TODO: The SDP MAY include an objectTags type that indicates the tag values of all the supported objects types.

TODO: define storage format as well as RTP payload format details.

8.1. Game State Tag Registry

The specification defines a new IANA registry for tag values. All values MUST be greater than zero.
Values 1-127 are assigned by "IETF Review" as defined in [RFC8126], and should only be used when size is critical, the object is small, and will be used frequently.

Values 127-16383 are assigned by "Specification Required" as defined in [RFC8126].

Values 16384 to 2,097,151 are "First Come First Served" as defined in [RFC8126].

Initial assignments are:

<table>
<thead>
<tr>
<th>TagName</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tagInvalid</td>
<td>0</td>
</tr>
<tr>
<td>tagHead1</td>
<td>1</td>
</tr>
<tr>
<td>tagHand1</td>
<td>2</td>
</tr>
<tr>
<td>tagObject1</td>
<td>3</td>
</tr>
<tr>
<td>tagParent1</td>
<td>4</td>
</tr>
<tr>
<td>tagMesh1</td>
<td>128</td>
</tr>
<tr>
<td>tagHand2</td>
<td>129</td>
</tr>
<tr>
<td>tagHeadIPD1</td>
<td>130</td>
</tr>
<tr>
<td>tagObject2</td>
<td>131</td>
</tr>
<tr>
<td>tagMesh2</td>
<td>132</td>
</tr>
</tbody>
</table>

Table 1

9. Security

Like most things in RTP, the data can be personal identifying information. For example, the Hand2 type of data when generated by tracking a persons hand might identify that user.

Appendix A. Acknowledgments

Thanks to Paul Jones for comments and writing an implementation.
Appendix B. Implementations

A C++ open source implementation is available at: TODO.

Appendix C. Test Vectors

C.1. Head Location

Head Location type 1 with head at location 1.1, 0.2, 30.0, no rotation (so quaternion 0, 0, 0, 1) and not rotating, an objectID of 4, a time of 5 ms after epoch and an IPD of 0.056.
Table 2

Appendix D. Encode API

API that take a high level representation of each object where types are all float or int and returns a memory buffer.

Appendix E. Decode API

API that takes binary data and set up objects and updates the objects and returns object ids that were updated.
For each object, an API to get the predicted values at a given time.

Appendix F.  EBNF

Head1 ::= tagHead1 Length ObjectID Time1 Loc2 Rot2
   ( tagHeadIpdl Length Float16 /* IPD */ )?

Mesh1 ::= tagMesh1 Length ObjectID
   ( TextureUrl1 | TextureRtpPT1 )
   VarUInt /* num Vertexes */
   Loc1+ /* vertexes */
   VarUInt /* numNormals */
   Norm1* /* normals */
   VarUInt /* numTextureCoord */
   TextureUV1* /* textureCoord */
   VarUInt /* numTrianglesIndex */
   VarUInt+ /* trianglesIndex */

Mesh2 ::= tagMesh2 Length ObjectID
   Loc2 Rot2 Scale2
   MeshUrl1
   ( TextureUrl1 | TextureRtpPT1 )? /* Optional Texture Map */
   ( tagParent1 Length ObjectID )? /* Optional Parent */

Object1 ::= tagObject1 Length ObjectID Time1
   Loc1
   Rot1
   Scale1
   ( tagParent1 Length ObjectID )? /* Optional Parent */

Object2 ::= tagObject2 Length ObjectID Time1
   Loc2
   Rot2
   Scale2
   ( tagParent1 Length ObjectID )? /* Optional Parent */

Hand1 ::= tagHand1 Length ObjectID Time1
   Boolean /* left */
   Loc2 Rot2

Hand2 ::= tagHand2 Length ObjectID Time1
   Boolean /* left */
   Loc2 Rot2
   Transform1 /* wrist */
   Transform1 /* thumbTip */
   Transform1 /* thumbIP */
   Transform1 /* thumbMCP */
   Transform1 /* thumbCMC */
Transform1 /* indexTip */
Transform1 /* indexDIP */
Transform1 /* indexPIP */
Transform1 /* indexMCP */
Transform1 /* indexCMC */
Transform1 /* middleTip */
Transform1 /* middleDIP */
Transform1 /* middlePIP */
Transform1 /* middleMCP */
Transform1 /* middleCMC */
Transform1 /* ringTip */
Transform1 /* ringDIP */
Transform1 /* ringPIP */
Transform1 /* ringMCP */
Transform1 /* ringCMC */
Transform1 /* pinkyTip */
Transform1 /* pinkyDIP */
Transform1 /* pinkyPIP */
Transform1 /* pinkyMCP */
Transform1 /* pinkyCMC */

Tag ::= VarUInt

tagInvalid ::= #x00
tagHead1 ::= #x01
tagHand1 ::= #x02
tagObject1 ::= #x03
tagParent1 ::= #x04
tagMesh1 ::= #x80 #x00
tagHand2 ::= #x80 #x01
tagHeadIpd ::= #x80 #x02
tagMesh2 ::= #x80 #x04
tagObject2 ::= #x80 #x03

ObjectID ::= VarUInt

Length ::= VarUInt

Loc1 ::= 
 Float32 /* x */
 Float32 /* y */
 Float32 /* z */

Loc2 ::= 
 Float32 /* x */
 Float32 /* y */
 Float32 /* z */
 Float16 /* vx */
Float16 /* vy */
Float16 /* vz */

Scale1 ::= 
Float16 /* all dimensions */

Scale2 ::= 
Float32 /* x */
Float32 /* y */
Float32 /* z */
Float16 /* vx */
Float16 /* vy */
Float16 /* vz */

Norm1 ::= 
Float16 /* x */
Float16 /* y */
Float16 /* z */

TextureUV1 ::= 
VarUInt /* u */
VarUInt /* v */

Rot1 ::= 
Float16 /* i */
Float16 /* j */
Float16 /* k */
/* w computed based on quaternion is normalized */

Rot2 ::= 
Float16 /* s.i */
Float16 /* s.j */
Float16 /* s.k */
Float16 /* e.i */
Float16 /* e.j */
Float16 /* e.k */

Transform1 ::= 
Float16 /* tx */
Float16 /* ty */
Float16 /* tz */

TextureUrl1 ::= String

MeshUrl1 ::= String

TextureRtpPT1 ::= UInt8 /* pt */
Tim1 ::= UInt16 /* time in ms */

Tag ::= VarUInt

Boolean ::= #x00 | #x01

String ::= VarUInt byte*

Blob ::= VarUInt byte*

Float16 ::= byte byte
Float32 ::= byte byte byte byte
Float64 ::= byte byte byte byte byte byte byte byte

Int8 ::= byte
Int16 ::= byte byte
Int32 ::= byte byte byte byte
Int64 ::= byte byte byte byte byte byte byte byte

UInt8 ::= byte
UInt16 ::= byte byte
UInt32 ::= byte byte byte byte
UInt64 ::= byte byte byte byte byte byte byte byte

VarUInt ::= ([#x0-#x7F] ) | ([#x80-#x87] byte ) | ([#x88-#x8B] byte byte ) | (#xE1 UInt32 ) | (#xE2 UInt64 )

VarInt ::= ([#x0-#x7F] ) | ([#x80-#x87] byte ) | ([#x88-#x8B] byte byte ) | (#xE1 Int32 ) | (#xE2 Int64 )

byte ::= [#x00-#xFF]

Authors’ Addresses

Cullen Jennings
cisco
Canada
Email: fluffy@iii.ca
Rich Logan
Cisco
United Kingdom
Email: rilogan@cisco.com