Transport Services API for WebRTC

Tommy Pauly
TAPS
IETF 103, November 2018, Bangkok

Background

WebRTC over QUIC proposal has inspired discussion around applying a transport-independent API

https://w3c.github.io/webrtc-quic/

Existing low-level APIs are transport-specific (RTCSctpTransport, RTCDtlsTransport, RTCIceTransport)

Raised Questions

Is Rendezvous sufficiently specified?

Do the states transitions match?

Does TAPS have any API gaps?

Are the data transfer models compatible?

Rendezvous

Resolve candidates *(RTCIceTransport)*

```
[]Preconnection := Preconnection.Resolve()
```

Establish Connections *(RTCQuicTransport/ RTCSCTPTransport)*

```
Preconnection.Rendezvous()
Preconnection -> RendezvousDone<Connection>
```

For established Connections on multi-streaming protocols, is the delivered Connection a specific stream?

State Transitions

Comparing TAPS to WebRTC proposal

TAPS States (Section 9)	TAPS Events (Section 11)	QUIC WebRTC States
Establishing	--	Connecting
Established	Ready	Connected
Closing	--	--
Closed	Closed	Closed
Closed	ConnectionError	Failed

API Gaps

Supporting Stop-Sending

WebRTC over QUIC proposes **`abortReading`**

A hard shutdown of the ReadableStream.

STOP_SENDING frame in QUIC

TAPS does not currently include this notion

When is this required as opposed to closing both with RST_STREAM+STOP_SENDING?

Data Transfer

TAPS provides Message semantics as well as Cloning connections for multiplexing

RTCQuicTransport presents a Stream abstraction

RTCSctpTransport presents a Data Channel abstraction

Data Transfer

Sending Data

TAPS

```
Connection.Send(
   messageData,
   messageContext,
   endOfMessage)
```

QUIC WebRTC

```
writable/writeBufferedAmount/writingAborted

write(buffer, length)
waitForWriteBufferedAmountBelow()
abortWriting()

dictionary RTCQuicStreamWriteParameters
{
   Uint8Array data;
   boolean finished = false;
};
```

Data Transfer

Receiving Data

TAPS

```
Connection.Receive(
   minIncompleteLength,
   maxLength)


Connection ->
   ReceivedPartial<
      messageData,
      messageContext,
      endOfMessage>
```

QUIC WebRTC

```
readable/readableAmount/readingAborted

readInto(buffer, length)->result
waitForReadable()
abortReading()

dictionary RTCQuicStreamReadResult
{
   unsigned long amount;
   boolean finished = false;
};
```