Media over QUIC (or BYOT)

At WebRTC TPAC 2018

People are already doing it!

- Live Media over HTTP over QUIC to MSE
  (Media and Entertainment Interest Group)
  They want better server push (QuicTransport)
- Live Media over SCTP to MSE
  https://blog.rainway.io/
  https://blog.parsecgaming.com/
  "I would be very interested in using QUIC instead of SCTP"
  "the weakest part of our implementation is SCTP"
- Live Media over websocket to/from WASM
  webrtchacks.com/zoom-avoids-using-webrtc/
  QuicTransport would be better than websocket

People can already do it (poorly)

	Audio	Video
Encode	WebAudio + WASM	Canvas + WASM
Decode	MSE or WebAudio + WASM	MSE or Canvas + WASM

People are planning on doing it

- Open Screen Protocol
- https://www.ietf.org/mailman/listinfo/rt-media-ng
- libwebrtc adding support for media over QUIC
  (for mobile; see slides at the end)
- experimental mobile client using libwebrtc indicates this can work, but:
  Web support is a problem which can be solved with the following proposals

Two orthogonal Proposals

1. Adopt QuicTransport
2. Add WHATWG streams to RtpSender/RtpReceiver for encoded media

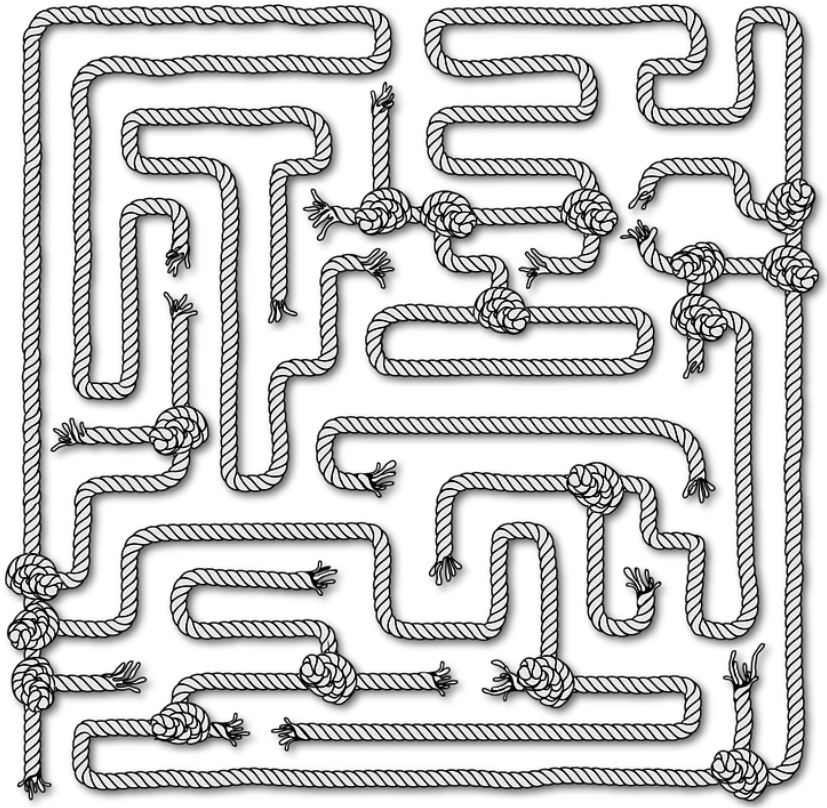
Proposal 1: Adopt QuicTransport

There is a lot of interest from outside the WG

We could continue work outside of the WG
(in ORTC CG and/or new incubation group)

Some people even prefer it to be outside
(Many people seem scared of anything with RTC prefix)

Does this WG want to be a part of this work?



Proposal 2: Add WHATWG streams to Sender/Receiver

```
interface mixin MediaSender {  // BYO transport
  ReadableStream readEncodedFrames(); // From encoder
  WritableStream writeFeedback(); // To Encoder (FIR, stats)
}

interface mixin MediaReceiver {  // BYO transport
  WritableStream writeEncodedFrames(); // To decoder
  ReadableStream readFeedback(); // From decoder (FIR, stats)
}

RtpSender includes MediaSender;  // Could make non-RTP sender later
RtpReceiver includes MediaReceiver;  // Could make non-RTP receiver
```

Example

```
var quic = ...;  // RTCQuicTransport
var sender = ...; // MediaSender (RtpSender for now; could be non-RTP later)
var receiver = ...; // MediaReceiver (RtpReceiver for now; could be non-RTP later)
var parser = transformStream(parseMessage);
var serializer = transformStream(serializeMessage);

// Send
quic.receiveStreams().pipeThrough(parser).pipeInto(receiver.writeEncodedFrames())
receiver.readFeedback().pipeThrough(serializer).pipeInto(quic.sendStreams());

// Receive
quic.receiveStreams().pipeThrough(parser).pipeInto(receiver.writeEncodedFrames())
receiver.readFeedback().pipeThrough(serializer).pipeInto(quic.sendStreams());
```

Proposal 3: BWE/RTT signal on QuicTransport

Surprise! A third proposal, but only if this WG adopts QuicTransport:

```
partial interface QuicTransport {
  // Naming scheme of https://wicg.github.io/netinfo/
  readonly attribute Megabit uplink;
  readonly attribute Millisecond rtt;
  attribute EventHandler onnetworkchange;  // uplink or rtt changed
}
```

In case you want to know what the chunks are

```
interface EncodedMediaFrame {
  attribute DOMTimeStamp timestamp;
  attribute DOMString mimeType;
  attribute SourceBuffer encodedData;
}

interface EncodedAudioFrame : EncodedMediaFrame {
  attribute unsigned short channels;
  attribute unsigned long samplesPerSecond;
  attribute unsigned long samples;
}

interface EncodedVideoFrame : EncodedMediaFrame {
  attribute DOMTimeStamp? duration;
  attribute bool keyFrame;
  attribute unsigned short rotation;  // in degrees
}
```

```
interface RTCKeyFrameRequest {
  ...
}

interface RTCMediaStats {
  ...
}
```

QUIC in libwebrtc (experimental)

We are already integrating QUIC transport in libwebrtc. We think that our work will also benefit web apps if we add proposed Web APIs:

- Easier to implement proposed API changes (Media Sender / Receiver) with in browsers that use libwebrtc
- Lower integration cost in applications (similar code-path in Web and native apps with QUIC).

Why QUIC for media and why we started this work:

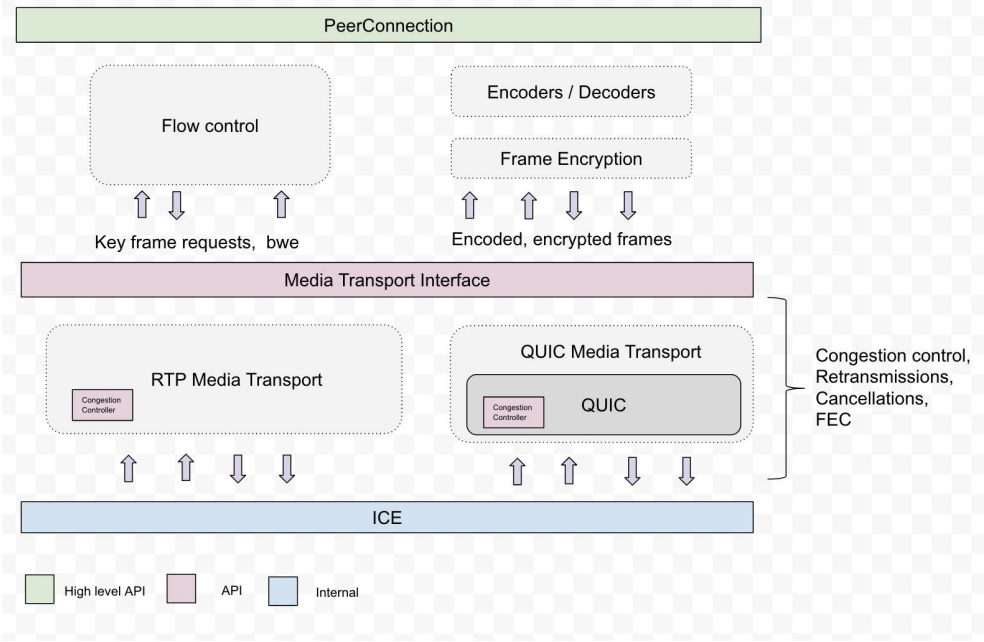
- Simple stack
- Fewer RTTs to setup encryption
- Opportunities for quality improvements (lower overhead, better retransmission control, easy to add additional frame metadata and control messages)
- Use QUIC for DataChannel with same congestion control as media

QUIC in libwebrtc (experimental)

We are refactoring webrtc codebase to support pluggable media transport

- RTP Transport
- QUIC Transport

Media transport interface = Send / Receive frames and key frame requests



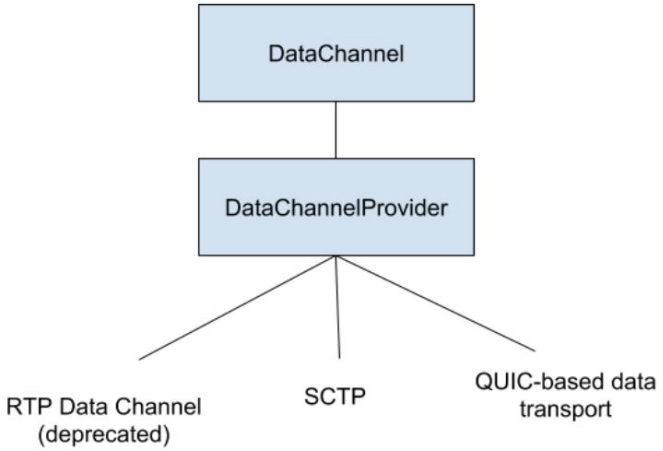
https://chromium.googlesource.com/external/webrtc/+/lkgr/api/media_transport_interface.h

QUIC in libwebrtc (experimental)

We are also implementing QUIC-based webrtc DataChannel,

- Applications can migrate to QUIC-based data channel without any changes.
- Media and Data will share congestion control

… but long term we want to adapt QuicTransport (both native and Web) and send media over QuicTransport instead of using QUIC directly.



QUIC in libwebrtc (experimental)

Media Transport is a thin layer between Encoders / Decoders and QuicTransport

- One QUIC stream per frame
  - Optional merge frames on low bitrate to reduce overhead
- Defines wire format for audio / video frames and control requests
  - Protobufs for serialization - easy, extendable
- Retransmission control
  - We rely on QUIC transmissions but configure retransmissions differently for audio / video
- Bandwidth allocation / adaptation
  - Currently above media transport in Sender / Receiver objects, but eventually can be moved to Media Transport and it will match proposed MediaSender / MediaReceiver APIs.

Pluggable component -- no need to standardize!!!

QUIC in libwebrtc (experimental)

Challenges

- Congestion control
  - BBR is not ready for real-time use (high initial delay during STARTUP, Insensitivity to latency buildups, Instability in bidirectional traffic flows)
  - We are working on integrating GoogCC from webrtclib into QUIC
  - Standardize GoogCC in QUIC? Fix BBR? Keep GoogCC separate (pluggable)? Web?
- SDP
  - Currently we assume that QUIC media transport will be negotiated by the App.
    How to make it work with PeerConnection?
    - Do not use PeerConnection APIs with QUIC media?
    - Cheat? Add “alternative media transport” option to SDP?
    - Add “quic media transport option” to SDP? But then we need to standardize protocol?
- FEC - not currently supported by QUIC
- What’s next for libwebrtc? PeerConnection or building blocks (ICE, media transport, encoders, decoders, etc)

Extra credit: For RTP E2EE

```
partial interface RtpSender {
  WritableStream writeEncodedFrames();  // To RTP transport
}

partial interface RtpReceiver {
  ReadableStream readEncodedFrames();  // From RTP transport
}
```

Example

```
var sender = ...; // RtpSender
var receiver = ...; // RtpReceiver
var encryptor = transformStream(encryptFrame);
var decryptor = transformStream(decryptFrame);

// Encrypt
sender.readEncodedFrames().pipeThrough(encryptor).pipeInto(sender.writeEncodedFrames())

// Decrypt
receiver.readEncodedFrames().pipeThrough(decryptor).pipeInto(receiver.writeEncodedFrames())
```