W3C, QuicTransport, and TAPS at IETF 103?
What a TAPS Web API might look like

callback Framer = ArrayBufferView (ArrayBufferView);
callback Deframer = ArrayBufferView (ArrayBufferView);

[Constructor(...)]
interface TapsPreconnection {
    void setFramer(Framer);  // Modulo Worklet complexity
    void setDeframer(Framer);  // Modulo Worklet complexity
    Promise<TapsConnection> initiate();
    Promise<TapsConnection> rendezvous();
    Promise listen();
    attribute EventHandler onconnection;  // TapsConnection
    void stop();
    attribute EventHandler onstopped;
}

interface TapsConnection {
    Promise<TapsWritableMessage> send(TapsSendParameters);
    Promise<TapsReadableMessage> receive(TapsReceiveParameters);
}

dictionary TapsSendParmaters {
    bool reliable = true;
    bool ordered = true;
    unsigned long lifetime;  // default: indefinite
    bool final = false;
    bool dontFragment = false;
    bool lowLatency = false;
}

dictionary TapsReceiveParameters {
    unsigned long? minLength = infinity;
    unsigned long? maxLength = infinity;
}
But there are some issues

● worklets are complicated
  ○ so application-controlled framer/deframer is complicated
● p2p is complicated
  ○ so you probably want to use RTCIceTransport
● Outside of p2p, you probably just want client->server (not act as a server on a web page)
  ○ so you probably need to take a URL like WebSocket does, which is guarded by CORS
● Everyone in the W3C wants WHATWG streams :)

Let's simplify a bit...
Take either RTCIceTransport or URL

callback Framer = ArrayBufferView (ArrayBufferView);
callback Deframer = ArrayBufferView (ArrayBufferView);

[Constructor(RTCIceTransport)]
[Constructor(DOMString url)]
interface TapsPreconnection {
    void setFramer(Framer);   // Modulo Worklet complexity
    void setDeframer(Framer);   // Modulo Worklet complexity
    Promise<TapsConnection> initiate();
    Promise<TapsConnection> rendezvous();
    Promise listen();
    attribute EventHandler onconnection;  // TapsConnection
    void stop();
    attribute EventHandler onstopped;
}

interface TapsConnection {
    Promise<TapsWritableMessage> send(TapsSendParameters);
    Promise<TapsReadableMessage> receive(TapsReceiveParameters);
}
Unify listen(), rendezvous(), and initiate()

callback Framer = ArrayBufferView (ArrayBufferView);
callback Deframer = ArrayBufferView (ArrayBufferView);

[Constructor(RTCIceTransport)]
[Constructor(DOMString url)]
interface TapsPreconnection {
    void setFramer(Framer);   // Modulo Worklet complexity
    void setDeframer(Framer);   // Modulo Worklet complexity
    Promise<TapsConnection> start();
    Promise<TapsConnection> rendezvous();
    Promise listen();
    attribute EventHandler onconnection;  // TapsConnection
    void stop();
    attribute EventHandler onstopped;
}

interface TapsConnection {
    Promise<TapsWritableMessage> send(TapsSendParameters);
    Promise<TapsReadableMessage> receive(TapsReceiveParameters);
}
callback Framer = ArrayBufferView (ArrayBufferView);
callback Deframer = ArrayBufferView (ArrayBufferView);

[Constructor(RTCIceTransport)]
[Constructor(DOMString url)]
interface TapsPreconnection {
  void setFramer(Framer);   // Modulo Worklet complexity
  void setDeframer(Framer); // Modulo Worklet complexity
  Promise<TapsConnection> start();
}

void stop();
  attribute EventHandler onstopped;
}

interface TapsConnection {
  Promise<TapsWritableMessage> send(TapsSendParameters);
  Promise<TapsReadableMessage> receive(TapsReceiveParameters);
}
Let the app do framing/deframing without worklets

```javascript
callback Framer = ArrayBufferView (ArrayBufferView);
callback Deframer = ArrayBufferView (ArrayBufferView);

[Constructor(RTCIceTransport)]
[Constructor(DOMString url)]
interface TapsConnection {
  void setFramer(Framer); // Modulo Worklet complexity
  void setDeframer(Deframer); // Modulo Worklet complexity
  Promise<TapsConnection> start();

  void stop();
  attribute EventHandler onstopped;

  Promise<TapsWritableMessage> send(TapsSendParameters);
  Promise<TapsReadableMessage> receive(TapsReceiveParameters);
}
```
Looks pretty good

[Constructor(RTCIceTransport)]
[Constructor(DOMString url)]
interface TapsConnection {

  Promise start();

  void stop();
  attribute EventHandler onstopped;

  Promise<TapsWritableMessage> send(TapsSendParameters);
  Promise<TapsReadableMessage> receive(TapsReceiveParameters);
}
interface TapsConnection {
  Promise start();
  Promise<TapsWritableMessage> send(TapsSendParameters);
  Promise<TapsReadableMessage> receive(TapsReceiveParameters);
  void stop();
  attribute EventHandler onstopped;
}
Add the message interfaces (with WHATWG streams)

```javascript
[Constructor(RTCIceTransport)]
[Constructor(DOMString url)]
interface TapsConnection {
    Promise start();
    Promise<TapsWritableMessage> send(TapsSendParameters);
    Promise<TapsReadableMessage> receive(TapsReceiveParameters);
    void stop();
    attribute EventHandler onstopped;
}

interface TapsWritableMessage {
    WritableStream writable;
    attribute EventHandler onexpired;
}

interface TapReadableMessage {
    ReadableStream readable;
    ... message context ...
}
 Compared to QuicTransport

interface TapsConnection {
    Promise start();
    Promise<WritableStream> send(TapsSendParameters);
    Promise<ReadableStream> receive(TapsReceiveParameters);
    void stop();
    attribute EventHandler onstopped;
}

interface TapsWritableMessage {
    WritableStream writable;
    attribute EventHandler onexpired;
}

interface TapReadableMessage {
    ReadableStream readable;
    ... message context ...
}

interface QuicTransport {
    void start(...);
    QuicStream createSendStream();
    attribute EventHandler onreceivestream; // QuicStream
    void stop();
    attribute EventHandler onstatechange;
    ...
}

interface QuicStream {
    readonly attribute ReadableStream? readable;
    readonly attribute WritableStream? writable;
}
They are very similar!

QuicTransport was designed as a low-level API on which higher-level APIs could be built.

A TAPS API could be built on top of QuicTransport.

But *just* a high-level API in the browser and not a lower-level API would give up QUIC-specific capabilities, such as:

- bidirectional streams
- DATAGRAMs (proposed extension)
- "RT streams" (proposed extension)
Missing from QuicTransport

There are some things missing from the QuicTransport:

- Ability to say "don't retransmit"
  - But it's been proposed
- Ability to say "use low-latency congestion control"
  - But it's been proposed as the default, at least when used with ICE (unclear for client/server case)
- Ability to say "don't fragment"
  - But it might make sense for QUIC DATAGRAM, if it's adopted