RTC Implementation Experiences

Justin Uberti, Google
IETF 107, March 2020
Typical HTTP-Based Application

- Client-server RPCs over HTTP
- May use media, but peer-to-peer
- Turn up on App Engine (or similar) in minutes
- Built-in authentication, failover, auto-scaling, metrics, logging
- AppRTC (first WebRTC app on App Engine) 1 eng, 1 week
Typical RTC Application

- Signaling over HTTP
- Media over SRTP/ICE
- Need to build own frontend service, FE/BE protocol, load balancing, authentication, failover, metrics
- Even need custom browser devtools
- Development can take dozens of eng-years
Typical RTC Protocol Stack

- Application/signaling over HTTP
- Video streamed over SRTP/ICE
- Real-time messaging streamed over SCTP/DTLS/ICE
- Hand-built servers, frontends, failover
- This is not a simple stack, on client or server
Someday?

- Ubiquity of HTTP + performance of WebRTC
- Build on H3 and WebTransport
- **Much simpler protocol stack**
- Seamless integration with HTTP server deployments
Why Standardize?

- WebTransport by itself isn't enough
  - Need to figure out how WebTransport is exposed in cloud providers
  - Need to figure out congestion control when WT is muxed with H3 traffic

- Want to allow interop between services
  - Any smart TV can consume a realtime media broadcast
  - Any cloud provider can ingest a realtime IP camera feed
What to Standardize?

1. Media over H3/WebTransport
   a. Media wire format (RTP over WT? QUIC streams?)
   b. Fallback mechanism for pre-H3

2. Some signaling mechanism over HTTP?
   a. RPC mechanism (e.g., REST or gRPC)
   b. RPC operations (e.g., start/stop stream)
   c. Media Descriptions (e.g., SDP or a replacement?)
   d. Negotiation (e.g., offer-answer or a replacement?)

#1 lets us build client APIs + cloud services that can easily produce + consume realtime media
#2 gives us cross-provider interop (e.g., send a stream from one provider to another)
Thanks