MHV24
February 11-14
Denver, CO, USA

Ultra-low Latency Video Delivery Over WebRTC Data Channels

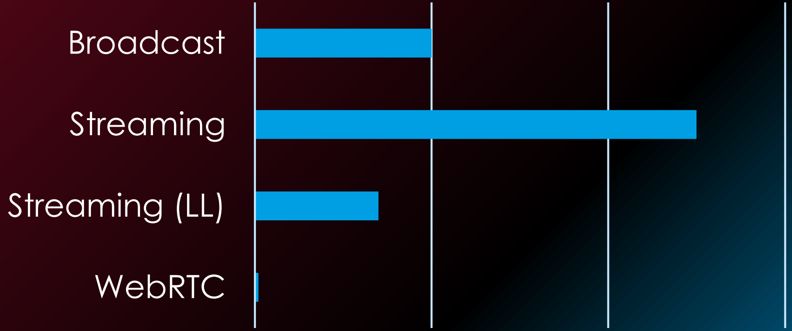
Nelson Francisco
Olie Baumann
Julien Le Tanou
Richard Fliam
MediaKind

Latency

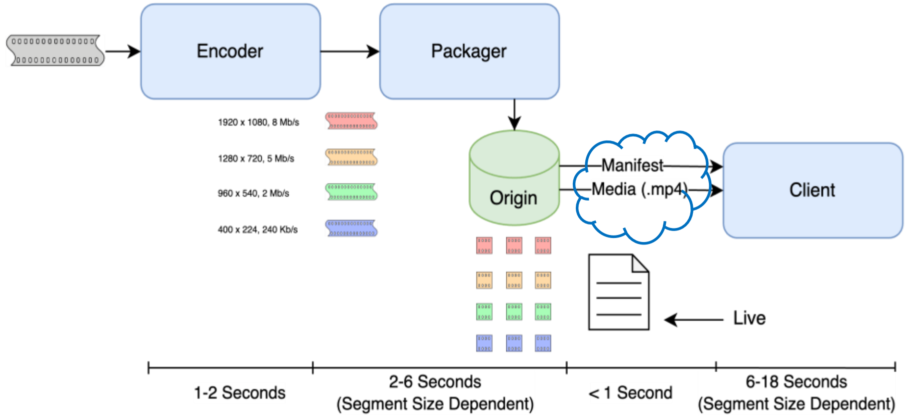
Video streaming industry has focused great interest in low-latency over the past few years:

- More variety and more heterogenous providers – race to be the first, preferred platform;
- Proliferation of social media brings almost instant updates to the viewers;
- Opportunities for monetization – spot betting;
- Increased variety of platforms where viewers consume video content.

Latency in Video Delivery



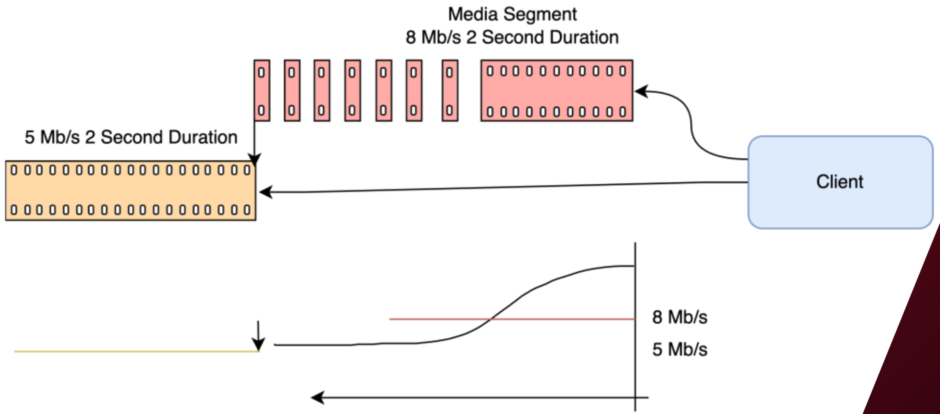
HTTP streaming



- Multiple renditions are made available at different resolutions and bitrates;
- For live streaming, the manifest is continually updated as new media segment are made available;
- The client chooses the bitrate that best suits its perceived network conditions.

- Delay on HTTP streaming is the result of the accumulated delay introduced by multiple elements in the delivery pipeline;
- Latency is closely linked to the segment delay;
- To lower the latency, segments can be split into chunks, written and made available more frequently – LL-HLS and LL-DASH.

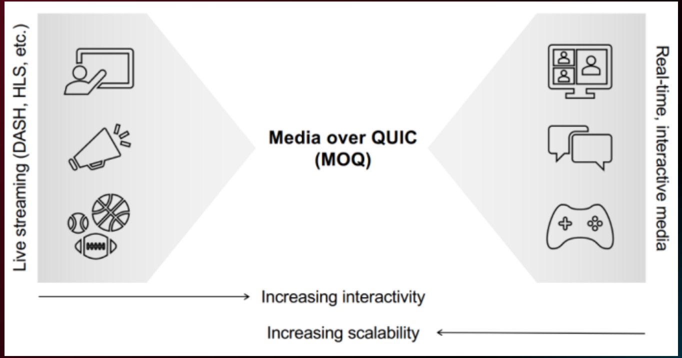
Limitation on HTTP streaming



- Simply reducing the segment length impacts video quality and increases CDN and server load;
- If the network becomes congested and packets are not getting through in time, they may be retried;
- Because packet delivery over TCP is reliable and ordered, all future packets are blocked until the lost one is successful – head-of-line blocking;
- HTTP does not allow "cancelling" the remainder of a segment if the CDN becomes congested;
- Slow to change renditions when network conditions change;
- Large buffers are required at the client side to guarantee a smooth playback when the delivery is bursty.

QUIC

- Developed by Google and standardized by the IETF (RFC9000);
- Media over QUIC (MOQ) working group formed in 2022 and tasked by IETF to study large-scale media transmission over QUIC;
- Need a custom application layer to reap all the benefits QUIC can provide at the transport layer;
- Still limited compatibility and support.

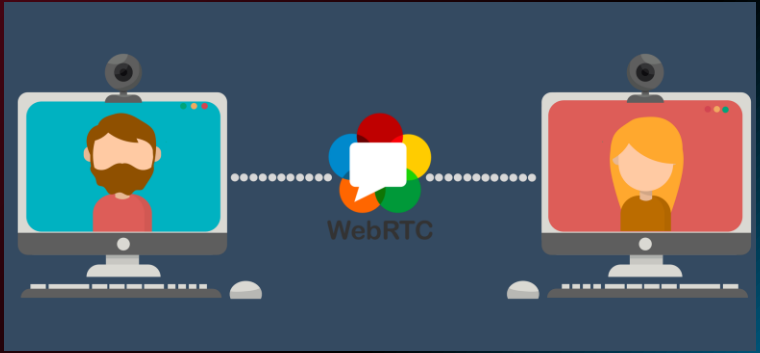


MOQ promises several advantages:

- Reduced connection establishment time;
- Transport Layer Security – encryption;
- Improved congestion control;
- Allows multiplexing of streams over a single connection to avoid HoL;
- Prioritization.

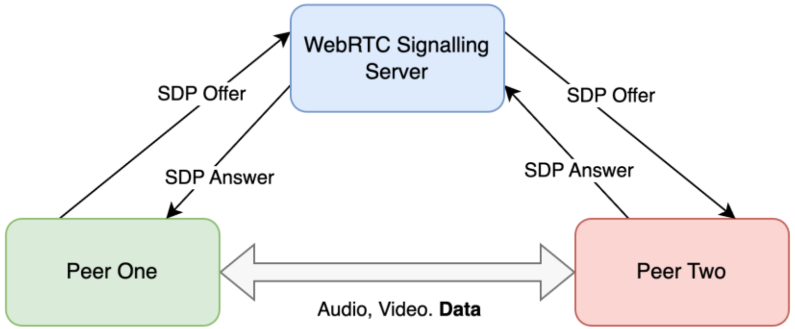
WebRTC

- Developed by Google c. 2010;
- Open-source protocol/libraries;
- Since WebRTC does not contain third-party software or plugins, it can pass through firewalls without losing quality or adding latency;
- Designed for bidirectional, real-time communication, WebRTC is the fastest protocol available with wide support;
- Primarily for Video Conferencing;
- Latencies in the 100s ms;
- UDP, not HTTP / TCP based;
- Supported by most browsers.



WebRTC

Establishing connection between two peers:



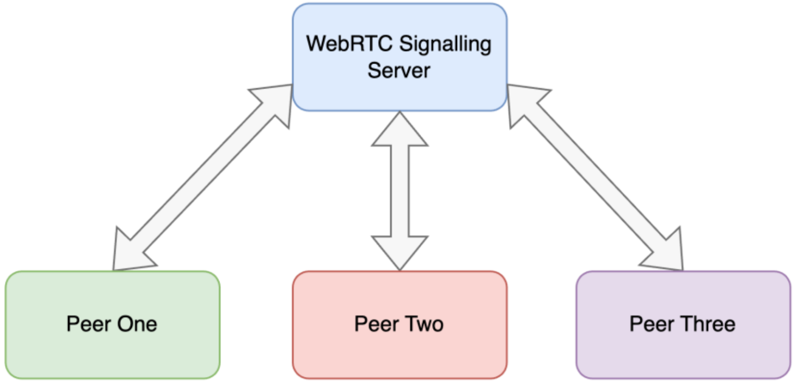
SDP Offer

- ICE Candidates
- Audio and Video Codecs

- Targets peer-to-peer applications such as video conferencing
- SPD offer contains the codecs Peer One wants to use and the address to contact Peer One directly
- Forwarded to Peer Two, who can refuse, put on hold or make counter-offer by making an SPD answer
- SPD answer forwarded to Peer One, and connection is established.
- As the network connection changes, the encoder adapts its bandwidth
- There are no multiple renditions to chose from.

WebRTC

WebRTC is not suited for one-to-many application scenarios:



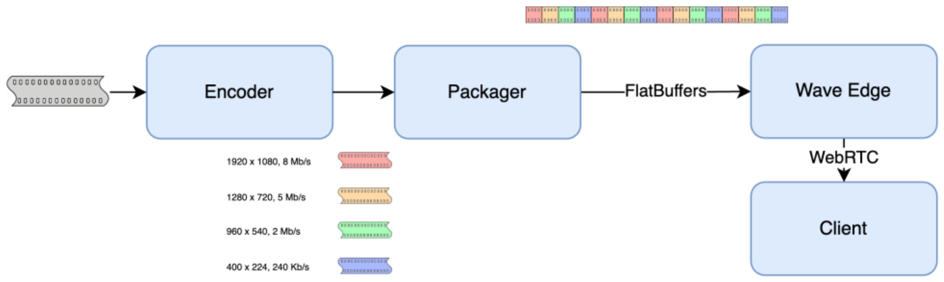
But WebRTC not suited for live streaming:

- Encode per client doesn't scale;
- Encoding tools in WebRTC are limited (low quality/bandwidth balance);
- No support for DRM.

However, by using WebRTC data channels, designed for arbitrary binary data:

- We can use a standard, broadcasting grade encoder;
- Control the video quality;
- Apply DRM;
- The pre-encoded and encrypted content can simply be replicated to all clients at the edge of the networkScale to millions of users.

WebRTC - the WAVE solution

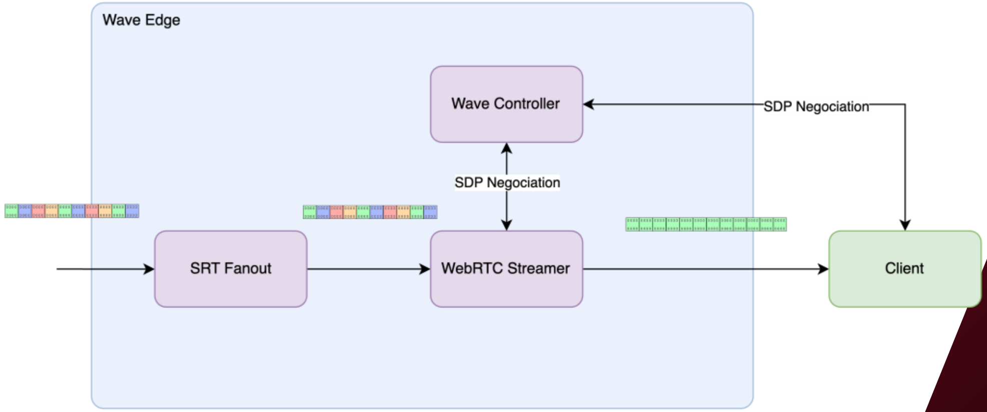


The WAVE solution can be split into three main sections:

- Encoding – use a standard encoder to generate all renditions;
- Distribution – serialize all renditions into a single buffer, and distribute to multiple WAVE Edge servers via SRT;
- Delivery – WAVE Edge servers deliver the different renditions to each client, using WebRTC data channels.

- The encoder produces all renditions;
- The packager segment the stream and serializes all the streams into a flatbuffer;
- At this stage, the content can be encrypted using standard DRM techniques;
- The regional Wave Edge receivers access the data in the flatbuffer via SRT;
- Each server then sends the data to multiple clients via WebRTC, deciding which of the received pre-encoded renditions to send to each client.

WAVE Edge

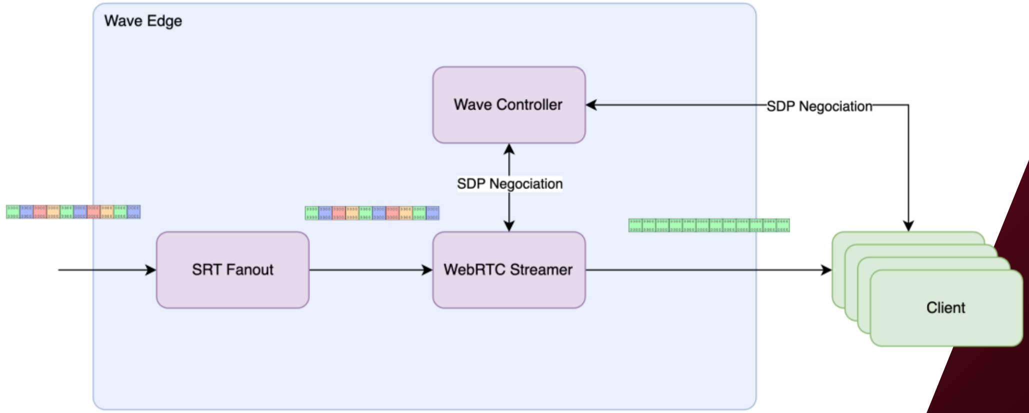


Kubernetes cluster containing multiple services:

- Wave controller (SDP negotiation and streamer scaling depending on the number of clients)
- SRT fanout
- WebRTC streamer

- The Wave controller manage the connections with the clients
- Feedback path from the client informs the streamer about network conditions
- The Streamer send to each client the best rendition for the bandwidth available

WAVE Edge

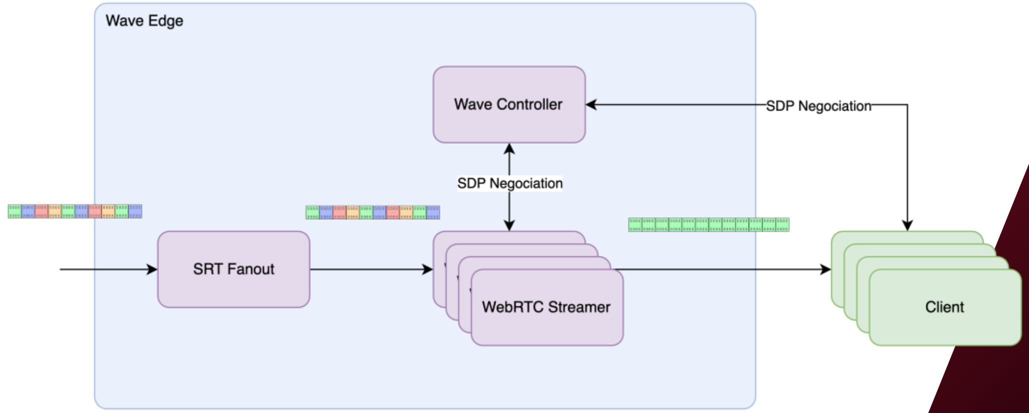


Kubernetes cluster containing multiple services:

- Wave controller (SDP negotiation and streamer scaling depending on the number of clients)
- SRT fanout
- WebRTC streamer

- The Wave controller manage the connections with the clients
- Feedback path from the client informs the streamer about network conditions
- The Streamer send to each client the best rendition for the bandwidth available
- Each Streamer handles many clients (limited by the node NIC throughput)
- It is very lightweight; in some use cases we are able to deploy the streamer on a home gateway

WAVE Edge



Kubernetes cluster containing multiple services:

- Wave controller (SDP negotiation and streamer scaling depending on the number of clients)
- SRT fanout
- WebRTC streamer

- The Wave controller manage the connections with the clients
- Feedback path from the client informs the streamer about network conditions
- The Streamer send to each client the best rendition for the bandwidth available
- Each Streamer handles many clients (limited by the node NIC throughput)
- It is very lightweight; in some use cases we are able to deploy the streamer on a home gateway
- More clients, more streamers - scaling is managed by the controller when the streamer node NIC is exceeded
- For more streamers, we need fanout to avoid excessive egress costs / replicating data locally rather than across regions

Implications of using Web RTC

➖
- WebRTC (SCTP/UDP) can be unreliable
- The profile must be chosen on the "server" side

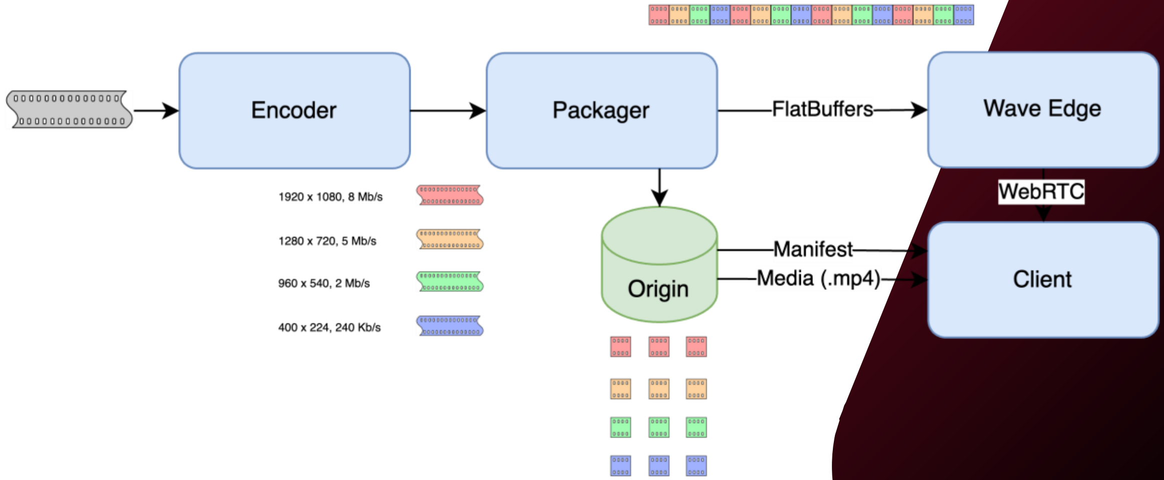
➕
- Forward Error Correction and interleaving manage the majority of losses
- Transitions to lower profiles can be quicker

➕
- Latency is a new ABR algorithm parameter
- WebRTC is all push, there is no buffer

➖
- ABR algorithms need to be rethought
- We still need HTTP for live pause and time shift

WebRTC plus HTTP

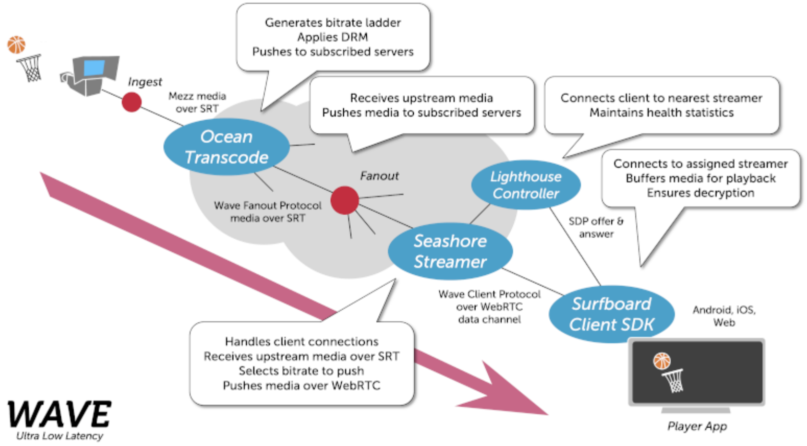
To handle replays, this ultra-low latency streaming solution is combined with a more traditional HTTP delivery mechanism:

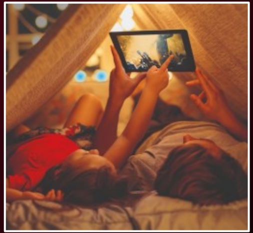


- The client can store content coming from the WebRTC stream to fill the gap between the live edge and the first reasonable playback point in the HLS playlist, typically twenty to thirty seconds.
- This way, consumers are able to navigate the seek bar in the normal way and return to the ultra-low latency feed at any time.

Summary

- WebRTC can be reliably used for ultra-low latency streaming
- By using Data Channels it is possible to:
  - Maintain control over picture quality
  - Apply standard DRM schemes
  - Scale to millions of clients


Generates bitrate ladder
Applies DRM
Pushes to subscribed servers
Receives upstream media
Pushes media to subscribed servers
Connects client to nearest streamer
Maintains health statistics
Connects to assigned streamer
Buffers media for playback
Ensures decryption
Ingest
Mezz media over SRT
Ocean Transcode
Fanout
Wave Fanout Protocol media over SRT
Lighthouse Controller
SDP offer & answer
Seashore Streamer
Wave Client Protocol over WebRTC data channel
Surfboard Client SDK
Android, iOS, Web
Player App
Handles client connections
Receives upstream media over SRT
Selects bitrate to push
Pushes media over WebRTC
WAVE
Ultra Low Latency



LIVE without limits

MediaKind