Frame/Stream Transport

Bernard Aboba
Microsoft Corporation
MoQ Virtual Interim
Tuesday, January 31, 2023

What We’ll Cover

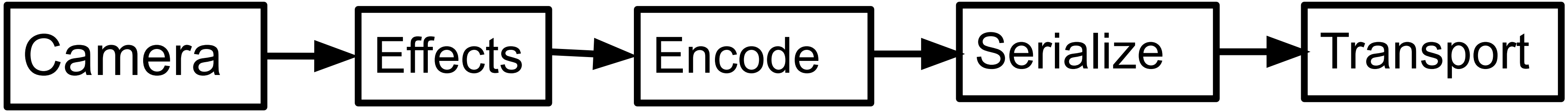
- Tools
  - Next Generation Web Media APIs
  - WebCodecs
  - WebTransport
- Frame/Stream Transport
  - Observations
  - Concurrency
  - Partial reliability

Next Generation Web Media APIs

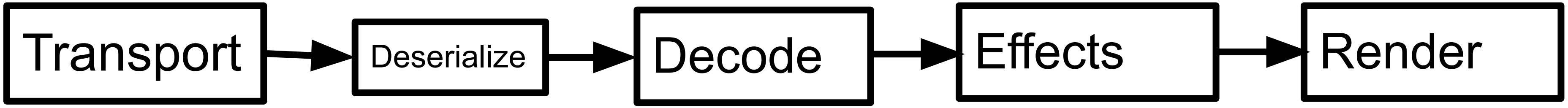
- Capture
  - Media Capture and Streams Extensions
  - Mediacapture-transform
- Discovery
  - Media Capabilities
- Encode/decode
  - WebCodecs (for raw media)
  - MSEv2 (for containerized media)
  - WebRTC-SVC (scalable video coding support, shipping in M111)
- Transport
  - WebTransport
  - WebRTC data channel in Workers
- Performance
  - Request VideoFrame Callback
- Framework
  - WHATWG Streams
  - Web Assembly

The “Pipeline” Model (WHATWG Streams)

- **Send**



- **Receive**



The “Pipeline” Model (in code)

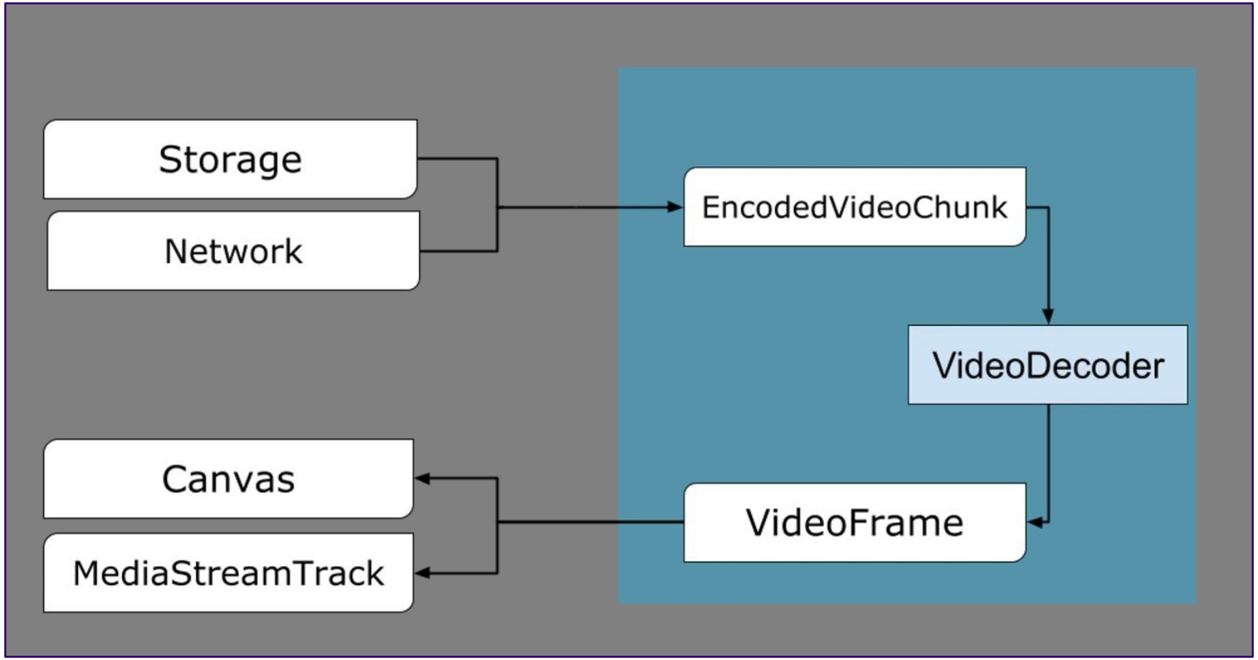
- Send

```
inputStream
    .pipeThrough(SpecialEffects())
    .pipeThrough(EncodeVideoStream(config))
    .pipeThrough(Serialize())
    .pipeTo(createSendStream())
```

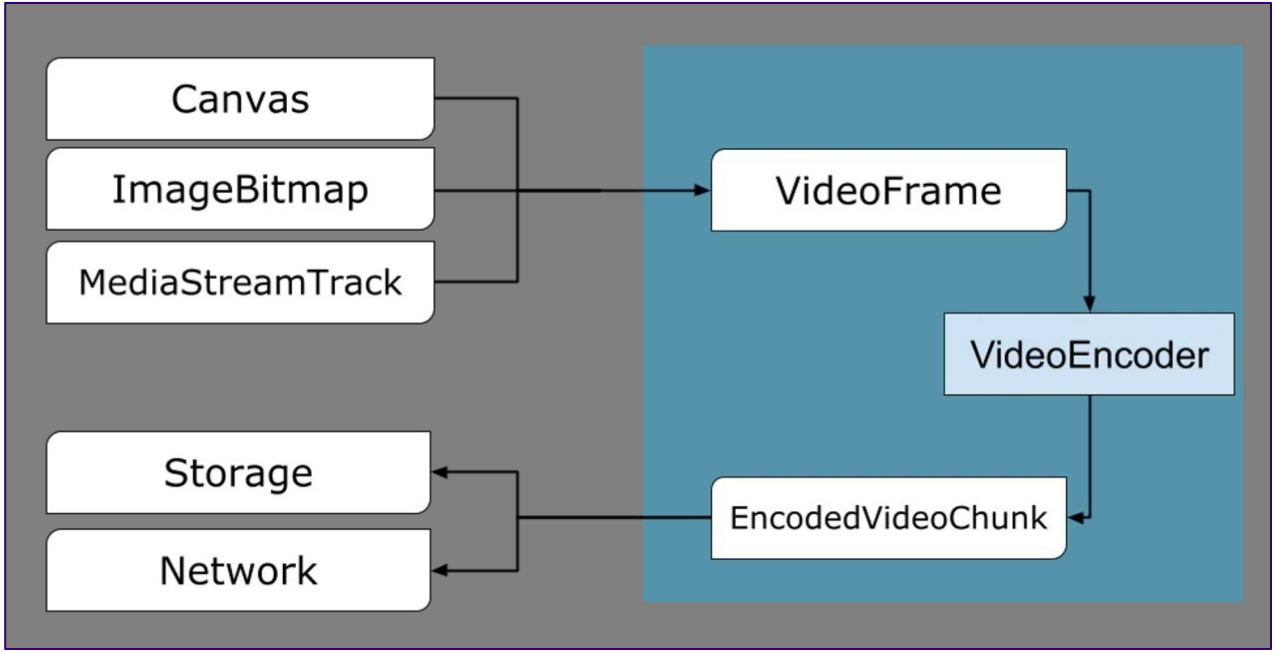
- Receive

```
createReceiveStream()
    .pipeThrough(Deserialize())
    .pipeThrough(DecodeVideoStream())
    .pipeTo(outputStream)
```

Video decoding in WebCodecs



Video decoding (similar just reversed)



WebCodecs Codec Support

For Chrome:

- `VideoDecoder`: AVC (H.264), VP8, VP9, AV1 and HEVC (hardware only)
- `AudioDecoder`: AAC, FLAC, MP3, Opus, Vorbis, μ-law and A-law PCM formats.
- `VideoEncoder`: H264, VP8, VP9, AV1 and HEVC (hardware only)
- `AudioEncoder`: Opus and AAC.

Use `isConfigSupported()`

Details of support can be platform / device specific

WebTransport Protocol Mappings (Section 10)

API Method	QUIC Protocol Action
writable.abort(errorCode)	sends RESET_STREAM with errorCode
writable.close()	sends STREAM_FINAL
writable.getWriter().write()	sends STREAM
writable.getWriter().close()	sends STREAM_FINAL
writable.getWriter().abort(errorCode)	sends RESET_STREAM with errorCode
readable.cancel(errorCode)	sends STOP_SENDING with errorCode
readable.getReader().cancel(errorCode)	sends STOP_SENDING with errorCode
wt.close(closeInfo)	terminates session with closeInfo

QUIC Protocol -> API Effect (Section 10)

QUIC Protocol Action	API Effect
received STOP_SENDING with errorCode	errors `writable` with `streamErrorCode`
received STREAM	(await `readable`.getReader().`read`()).value
received STREAM_FINAL	(await `readable`.getReader().`read`()).done
received RESET_STREAM with errorCode	errors `readable` with `streamErrorCode`
Session cleanly terminated with closeInfo	(await wt.`closed`).closeInfo, and errors open streams
Network error	(await wt.`closed`) rejects, and errors open streams

RVFC Timing Model

§ 2. VideoFrameCallbackMetadata

```
dictionary VideoFrameCallbackMetadata {
  required DOMHighResTimeStamp presentationTime;
  required DOMHighResTimeStamp expectedDisplayTime;

  required unsigned long width;
  required unsigned long height;
  required double mediaTime;

  required unsigned long presentedFrames;
  double processingDuration;

  DOMHighResTimeStamp captureTime;
  DOMHighResTimeStamp receiveTime;
  unsigned long rtpTimestamp;
};
```

For More Information

Tutorials

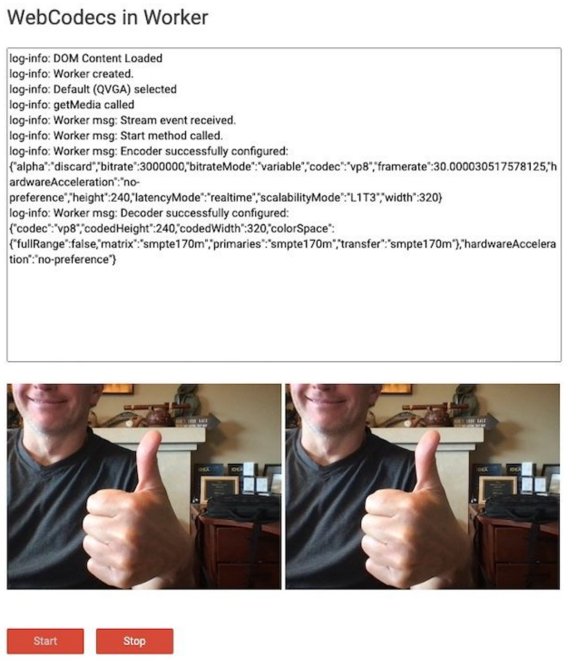
- WebCodecs
- WebTransport

W3C Web Media Pipeline Architecture Repo

- Sample code
- Architecture Issues
- Links to specs

Two Samples

1. Sample #1 encodes and decodes video in a WHATWG Streams pipeline without transport.
   a. Live site: https://webrtc.internaut.com/wc/wcWorker/
   b. Github repo: https://github.com/aboba/wc-demo/
2. Sample #2 adds network transport to the sending and receiving pipelines, bouncing encoded frames off a relay in the cloud. Comparison with sample #1 can help isolate network effects.
   a. Live site (Chrome Stable): https://webrtc.internaut.com/wc/wtSender10/
   b. GitHub repo: https://github.com/aboba/wt-demo



What’s in the Samples

- WHATWG Streams-based Receive and send pipelines.
  - Send and receive pipelines in a (single) worker.
    - Transferable streams used to tunnel video to/from the main thread.
  - Encode/Decode stages based on WebCodecs
  - Send/Receive transport based on WebTransport frame/stream transport (no datagrams).
    - Uncontainerized (raw) video.
  - Conversion from VideoFrames <-> MediaStreamTracks via Mediacapture-transform API.
  - Frames bounced off a store & forward relay.
    - To do: cut-through relay.
- Partial reliability.
  - Used along with Scalable Video Coding (temporal scalability).
  - ‘RESET_FRAME timer’ set lower for discardable (extension layer) frames. Base layer frames considered ‘non-discardable’.
- Concurrency:
  - Send pipeline: multiple frames in transit
    - P-frames sent alongside (much larger) I-frames
  - To do: fully concurrent reading

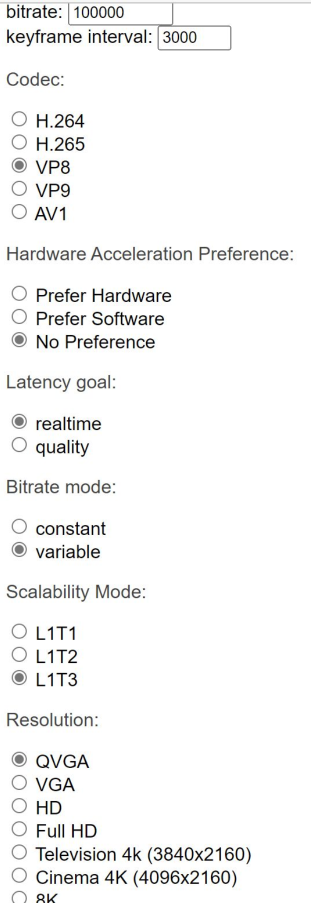
Frame/Stream Transport

- Send pipeline
  - Sender opens a uni-directional stream.
  - Sender sets a timer with an expiration (RTO)
    - If the timer fires, sender resets the stream
    - RTO values can differ between discardable and non-discardable frames.
  - Sender writes the header + payload to the uni-directional stream
  - Sender closes the stream.
- Receive pipeline
  - Receiver is notified of an incoming uni-directional stream.
  - Receiver reads from incoming uni-directional streams until:
    - A stream is closed OR
    - The stream is reset.
    - Length field helpful for memory allocation as well as to check the frame was completely received.

What You Can Do

- Vary encoding parameters, codecs, bitrates, resolutions, etc.
  - Can visually compare local and remote video
  - Can (manually) measure glass-glass latency
  - Can find bugs and gasp in horror!
    - VP9 + SVC + VBR = Boom!
- Post-experiment diagnostics
  - Metrics: RTT stats, loss, reordering, etc.
    - Calculated at application layer
    - Chrome does not surface QUIC stack metrics yet
  - Graphs: RTT versus frame length
  - To do: latency breakdown in each pipeline stage

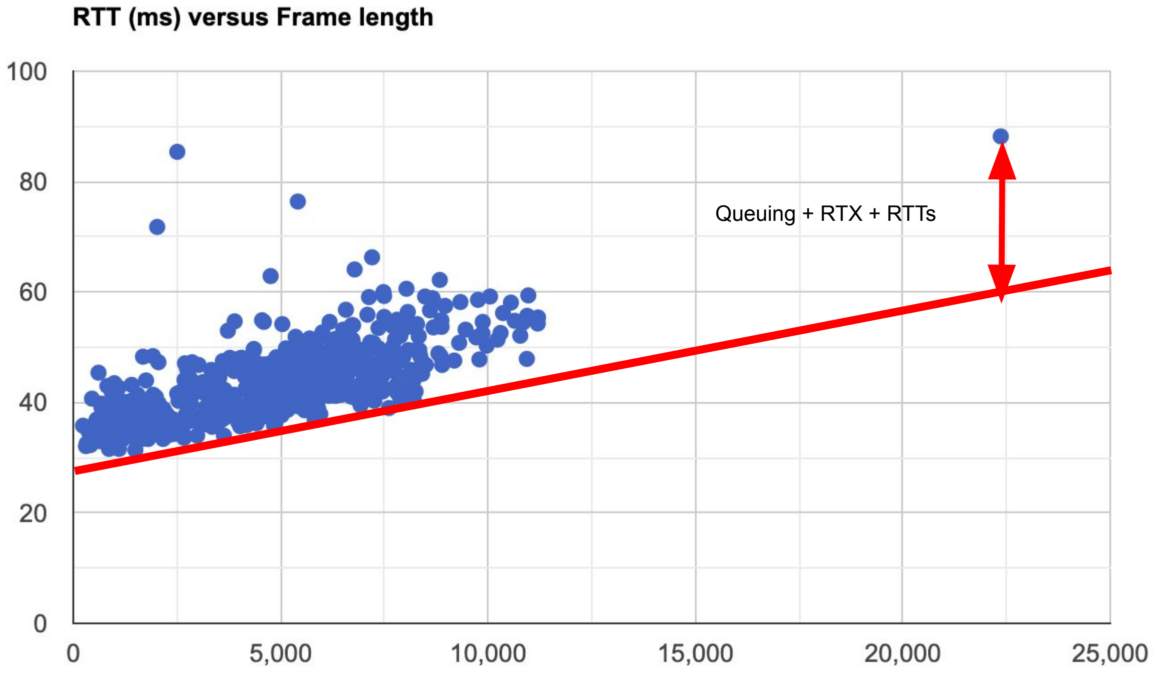
Parameters to Select



- Bitrate: “Average Target Bitrate” target provided to the encoder.
- Keyframe interval: number of frames between each keyframe.
- Codec: H.264, H.265, VP8, VP9 or AV1
  - H.265 support only where hw acceleration is available.
- Hardware Acceleration Preference: require hw acceleration, require sw only or “no preference”. Hw acceleration often not available.
- Latency goal: “quality” produces smaller frame sizes, but takes (marginally) longer than “realtime”.
- Bitrate mode: Constant Bitrate (CBR) or Variable Bitrate (VBR).
- Scalability mode: how many temporal layers to use. Enables differential protection for the base layer.
- Resolution: reflected in getUserMedia constraints. If your camera doesn’t support the requested resolution, window will be blacked out.

Example

- AV1 @ full-Hd with 1152.4 Kbps average bitrate and 30 fps, GoP = 3000, L1T3 scalability mode
- Largest (I-)frame = 22361 octets, median (P-)frame size = 4894 octets
- RTTmin = 31.4 ms. RTTmax = 88.2 ms
- I-frame further from the transmission line than vast majority of P-frames.
    - I-frames most likely to experience loss, queuing delays, multiple RTTs if cwind < 22361



BWE report:

{"count":553,"loss":0,"reorder":1,bwu":1152379.52,"seqmin":0,"seqmax":552,"lenmin":234,"lenfquart":2715.5,"lenmedian":4894,"lentquart":6605.5,"lenmax":22361,"recvsum":2657416}

RTT report:

{"count":553,"min":31.4,"fquart":38.6495,"avg":43.41247377938517,"median":42.1,"tquart":47.301,"max":88.199,"stdev":7.155866034173623,"srtt":40.33911687334871,"rttvar":2.802225680520845,"rto":51.54801959543209}

High Level Observations

- Video quality
  - AV1 can encode/decode in full-HD on new hardware, producing passable video quality even at low bitrates (<300 Kbps)
- Resilience
  - WebTransport frame/stream + temporal scalability provides excellent resilience.
  - Most losses due to RESET_FRAME timeouts (e.g. discardable frames)
- RESET handling
  - RESET_FRAMEs SHOULD be surfaced immediately (but don’t always appear to be).
    - Frames can be received intact, even after a RESET_FRAME is sent!

High Level Observations (cont’d)

- P-frames are typically small (a few packets) and exhibit frame RTT clustered around the “transmission line”.
- I-frames are ***much*** larger (10X or more) and often exhibit frame RTT considerably above the “transmission line”.
    - Effect most pronounced with high GoP sizes and low concurrency.
    - Effect seen even for low bandwidth utilization and low loss.
        - Suggests this is not just due to queuing delay or retransmissions
        - If frame size > cwind, requires multiple roundtrips.
    - Observation: increased concurrency lowers the gap between the frame RTT and the transmission line.

Thoughts on Concurrency
https://webrtc.internaut.com/wc/wtSender10/

- Concurrency desirable (lowers glass-glass latency) but implies more open streams.
  - For concurrency, both send and receive pipelines need to avoid blocking.
  - In Javascript, be wary of **await!**
    - `promise.then(f).catch()` is ***not*** the same as **await f**!
- A ***good*** sign: multiple P-frames arrive on the receiver prior to complete receipt of the initial I-frame.
  - Lack of re-ordering is a ***symptom of blocking!***
- Greater concurrency in frame/stream transport implies more re-ordering
  - More P-frames will be sent concurrently with initial (and subsequent) I-frames
  - Moves I-frame RTT closer to transmission line (cwind grows faster)
  - See `async writeChunk()` function implementing frame/stream sender
  - More work needed on read pipeline

Thoughts on Partial Reliability

- Temporal scalability enables “differential transport” on the sender.
  - Sender can set timer, send RESET_STREAM if timer expires.
  - Timer set lower for “discardable” frames (extension layers).
  - High timer set for “non-discardable” frames (base layer).
- Implications
  - Relay needs to forward RESET_STREAM frames.
    - Relay can receive RESET_STREAM before or after FIN.
  - Receiver needs to be prepared for loss of the RESET_STREAM frame.
    - Useful to have a Length field at the beginning of the packet.
    - Helps receiver to verify that it has received the complete frame.
    - Length field needs to be large enough for high resolution I-frames (could be 200KB+)