Joint MEDIA & WebRTC WG Meeting

September 15, 2023
14:30 - 16:30 Seville time
5:30 - 7:30 Pacific Time
12:30 - 14:30 UTC

W3C WG IPR Policy

- WebRTC WG and Media WG abide by the W3C Patent Policy: https://www.w3.org/Consortium/Patent-Policy/
- Only people and companies listed at https://www.w3.org/groups/wg/webrtc/ipr/ and https://www.w3.org/groups/wg/media/ipr/ are allowed to make substantive contributions to the specs

W3C Code of Conduct

- This meeting operates under W3C Code of Ethics and Professional Conduct

- We're all passionate about improving the Web, but let's all keep the conversations cordial and professional

Safety Reminders

While attending TPAC, follow the health rules:

- Authorized masks are required indoors at all times. If you need to remove your mask during the meeting, please keep it short
- Daily test is expected

Please be aware of and respect the personal boundaries of your fellow participants

https://www.w3.org/2023/09/TPAC/health.html

Welcome!

- Welcome to the September 2023 joint meeting of the W3C MEDIA and WEBRTC WGs, at which we will cover:
  - Introduction, QP rate control, hardware encode/decode errors, new encode/decode APIs.

About TPAC 2023 Meetings

- TPAC 2023 Schedule
- Meeting info:
  - https://www.w3.org/2011/04/webrtc/wiki/September_15_2023
- Link to slides has been published on WG wiki
- Scribe? IRC http://irc.w3.org/ Channel: #webrtc
- Will we be recording the session?
- Volunteers for note taking?

Virtual Meeting Tips (Zoom)

- **Both local and remote participants need to be on irc.w3.org channel #webrtc.**
- **Use “raise hand” to get into the speaker queue and “lower hand” to get out of the speaker queue.**
- **To try out WebCodecs over RTCDatachannel (not RTP!) join using a Browser.**
- **Please use headphones when speaking to avoid echo.**
- **Please wait for microphone access to be granted before speaking.**
- **Please state your full name before speaking.**

Agenda

- 14:40 - 14:55 Introduction
- 14:55 - 15:10 Frame QP-based rate control demo (Eugene)
- 15:10 - 15:25 Hardware encoding/decoding error handling (Bernard & Fippo)
- 15:25 - 16:25 New Encoder API (Erik)
- 16:25 - 16:30 Wrapup and Next Steps (Chairs)

Time control:

- A warning will be given 2 minutes before time is up.
- Once time has elapsed we will move on to the next item.

Introduction

Start Time: 14:40

End Time: 14:55

Background

- Streaming and RTC use cases are converging. The WebRTC-Extended Use case document now includes:
  - Section 3.2.1: Game streaming
  - Section 3.2.2: Low latency broadcast with fanout
- These new use cases require both low/ultra low latency and massive scale, raising issues in both WebCodecs and WebRTC.
- Today, these issues are often solved separately in both specifications and the code base, raising concerns about duplication of effort:
  - JS: WebRTC Encoded Transform often used as a “Poor man’s WebCodecs”
  - JS: Adding low-level controls to a high-level API (WebRTC)
  - Two distinct C/C++ encode/decode APIs

Examples of Similar Issues

- Rate control
  - QP-based rate control in WebCodecs (Eugene)
  - Issue 1234020: Develop H264 software rate controller usable from hardware encoders
- Performance/Hardware acceleration
  - Hardware encode/decode error handling (Bernard & Fippo)
  - Issue 384: HDR support (WebCodecs)
  - Issue 1351638: VideoEncoder encode times are inconsistently too long on Intel CPUs
  - VideoDecoder decode times consistently too long for some codecs/CPUs
- Codec support
  - HEVC
    - https://bugs.chromium.org/p/webrtc/issues/detail?id=13485 (HEVC RTP payload)
    - Issue 543: WPT tests lack HEVC (WebCodecs)
  - AV1 screen content coding:
    - https://bugs.chromium.org/p/webrtc/issues/detail?id=13929 (AV1 SCC in WebRTC)
    - https://bugs.chromium.org/p/chromium/issues/detail?id=1464862 (AV1 SCC in WebCodecs)
- Support for advanced video (SVC, simulcast)
  - Issue 44: Simulcast support (WebCodecs)
  - Issue 619: Inconsistent SVC metadata between WebCodecs and WebRTC Encoded Transform
  - Issue 285: Reference frame control (WebCodecs)
  - Issue 13: RPSI RTCP feedback support (WebRTC)

Is there a better way forward?

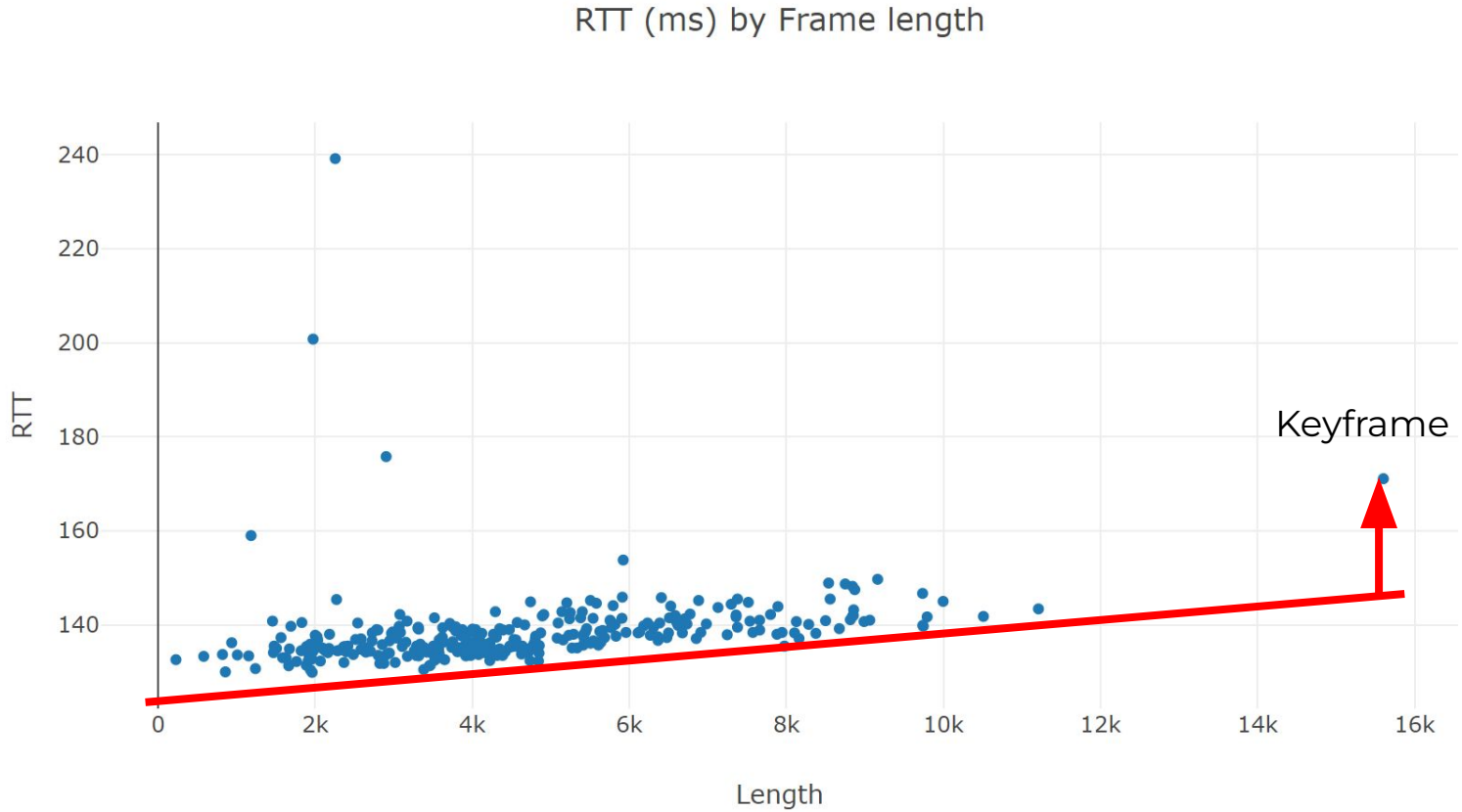
- Is the goal to enable WebRTC to support every desirable feature?
  - Or is it to enable applications to build their own support?
- This week, at the WEBRTC WG meeting, we have explored:
  - Combining WebCodecs with RTPTransport, to enable:
    - Custom RTP packetization (examples:. AAC, HEVC)
    - Large metadata
    - Support for custom robustness and feedback
- Today we will explore:
  - Common issues encountered in both WebCodecs and WebRTC
  - A proposal for a unified encoder API (Erik)

WebCodecs-WebTransport Echo

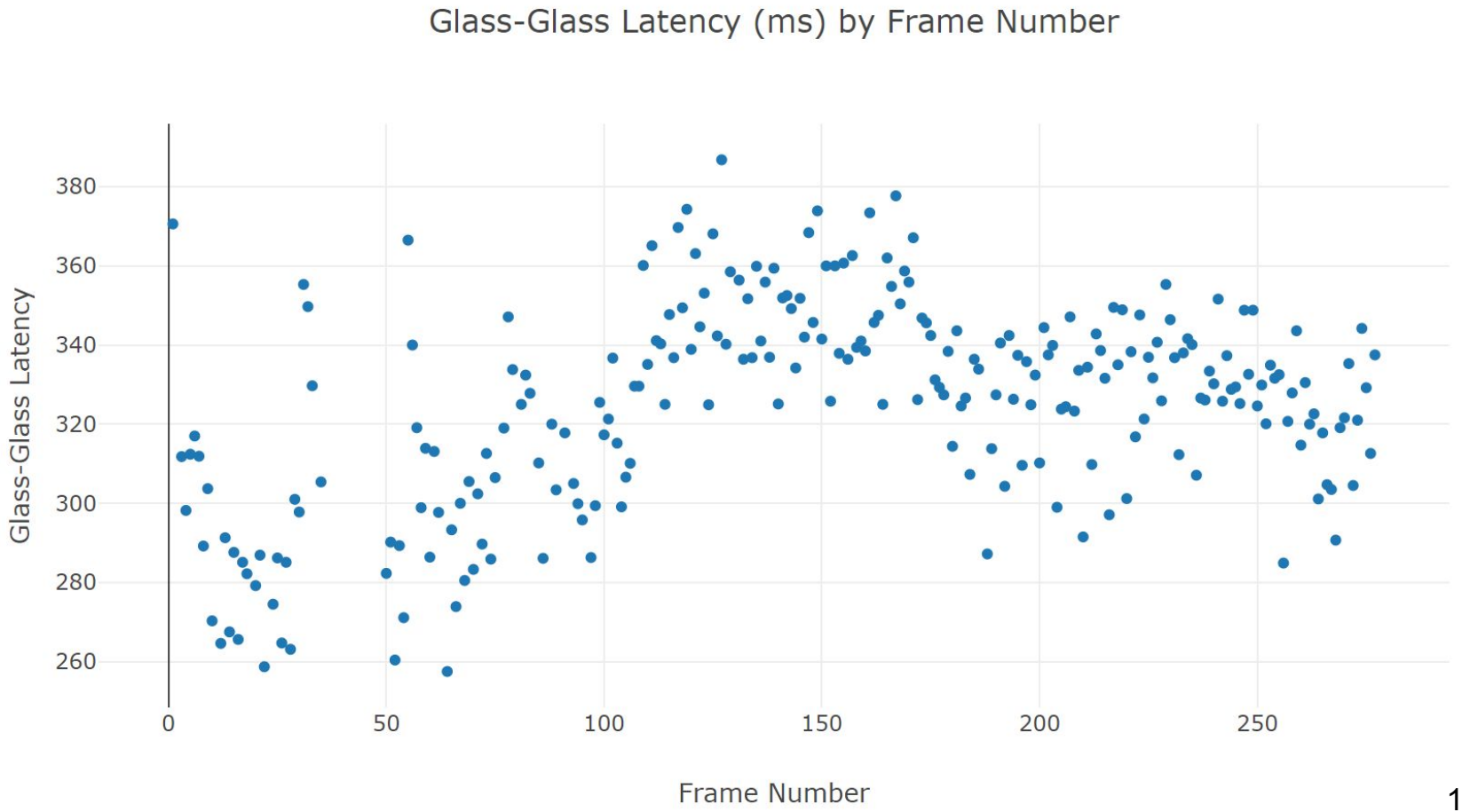
- GitHub repo: https://github.com/aboba/wt-demo
- Live demo: https://webrtc.internaut.com/wc/wtSender13/
- Functionality added since TPAC 2022
  - Added more controls (bitrate mode, decoder hardware preference)
  - Addressed race conditions
  - Use of plotly graphing library instead of Google graphs (Fippo)
    - Additional data provided when hovering over a data point
  - Automated measurement of glass-glass latency using the RVFC API:
  - https://wicg.github.io/video-rvfc/
  - Graphs of glass-glass latency, frame RTT, encode and decode latency
- Some (surprising?) observations
  - For higher resolutions, glass-to-glass latency >> frame RTT
  - Decoding latency can be a significant contributor (larger than encoder latency!)
    - Not observed for all codecs; needs more investigation

Example

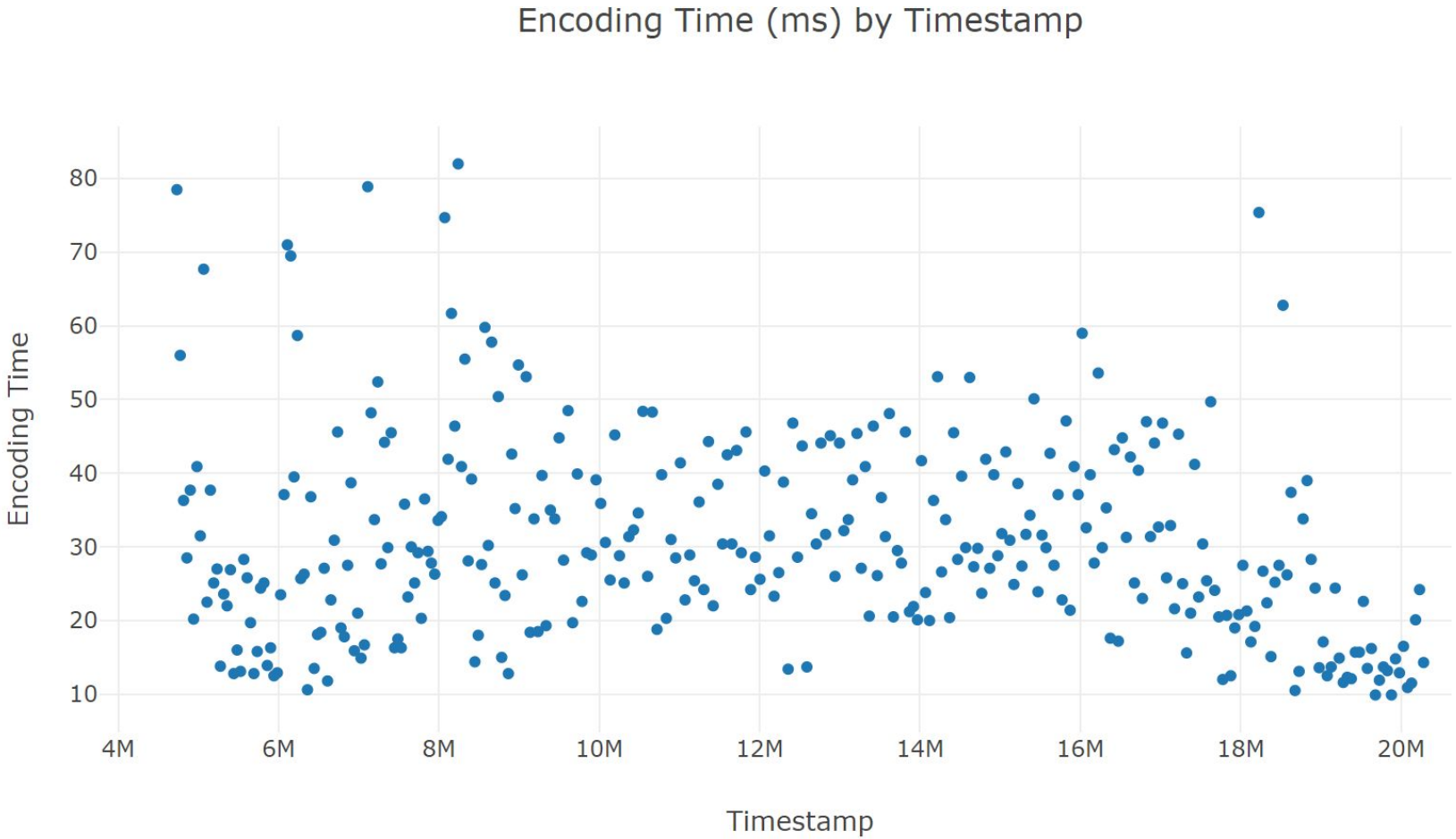
Configuration:
{"alpha":"discard","bitrate":1000000,"bitrateMode":"variable","codec":"av01.0.08M.10.0.110.09","framerate":30,"hardwareAcceleration":"no-preference","height":1080,"latencyMode":"realtime","scalabilityMode":"L1T3","width":1920}



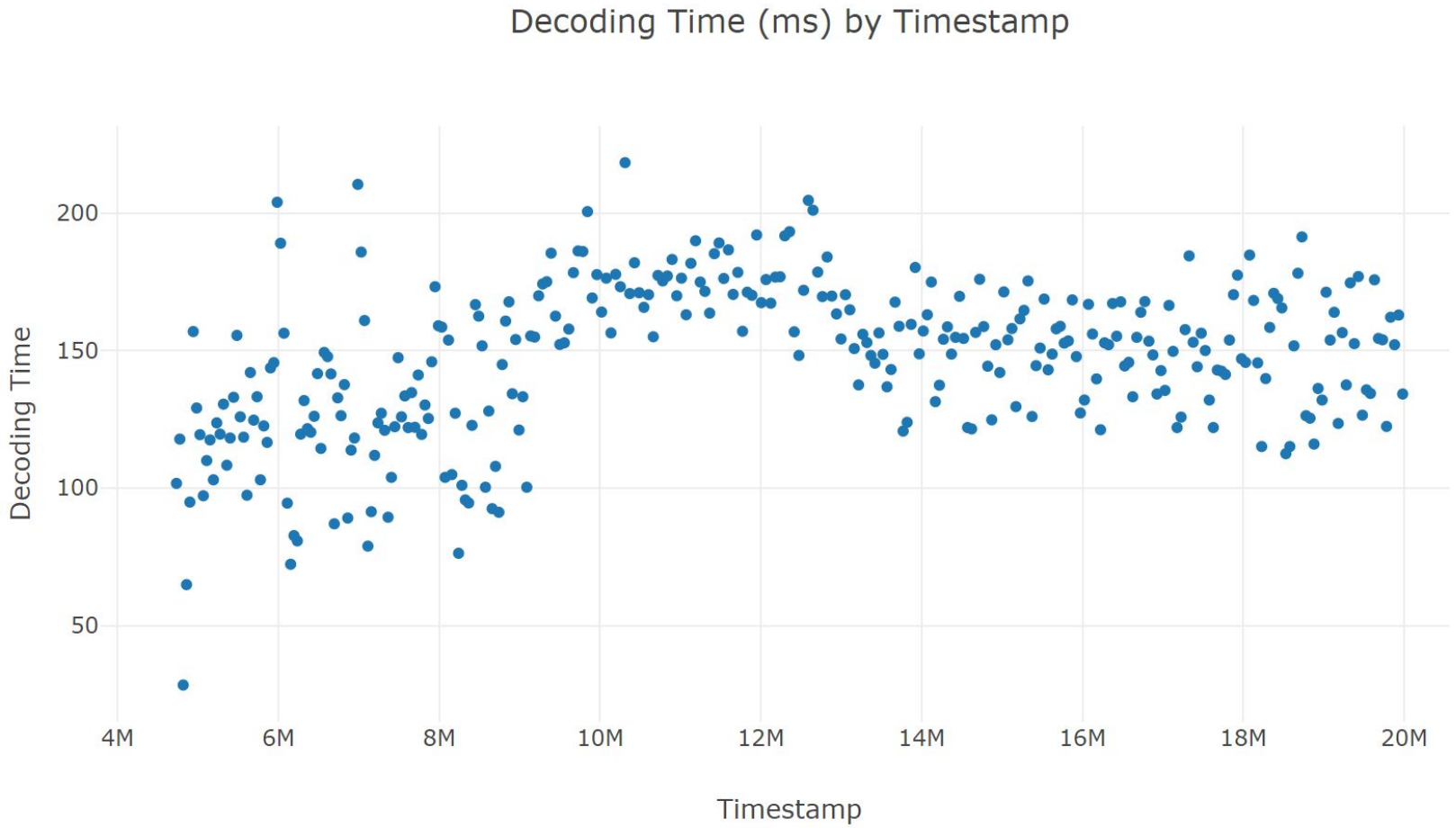
Glass-Glass Latency



Encoding Latency



Decoding Latency



Discussion (End Time: 14:55)

-

QP-based Rate Control (Eugene)

Start Time: 14:55

End Time: 15:10

Frame QP-based Rate Control in WebCodecs

```
enum VideoEncoderBitrateMode {
  "constant",
  "variable",
  "quantizer"
};
```

constant

Encode at a constant bitrate. See bitrate.

variable

Encode using a variable bitrate, allowing more space to be used for complex signals and less space for less complex signals. See bitrate.

quantizer

Encode using a quantizer, that is specified for each video frame in codec specific extensions of VideoEncoderEncodeOptions.

VideoEncoderEncodeOptionsForAv1

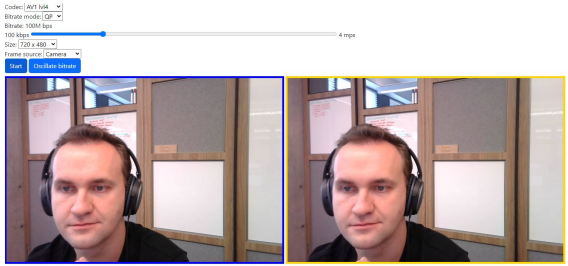
```
dictionary VideoEncoderEncodeOptionsForAv1 {
  unsigned short? quantizer;
};
```

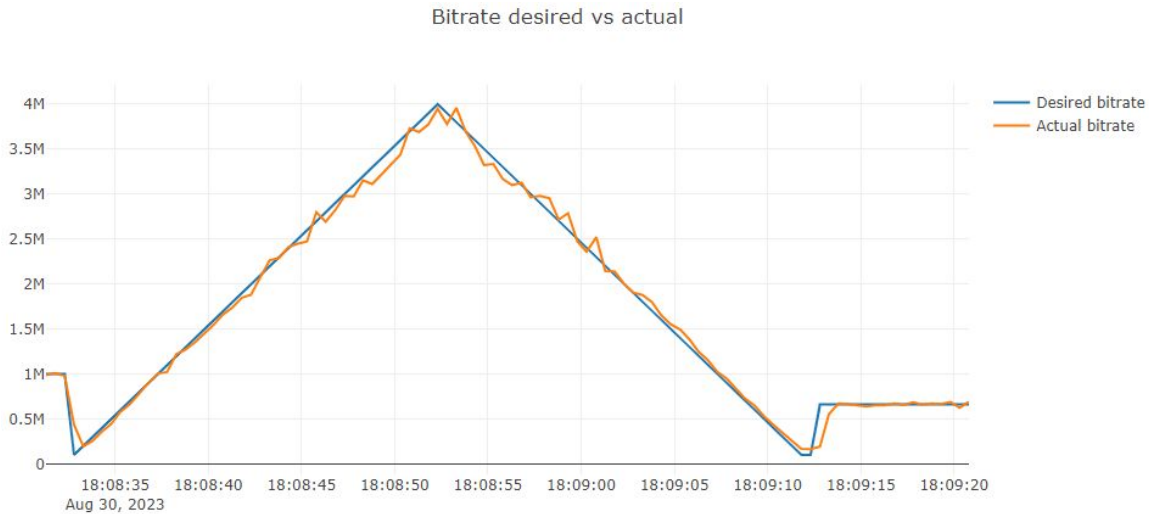
quantizer, of type unsigned short, nullable

Sets per-frame quantizer value.

QP-based Rate Control Demo

https://webcodecs-qp.glitch.me/





Primitive QP adjustment algorithm

```
const frames_to_consider = 4;
const frame_budget_bytes = (this.bitrate / this.fps) / 8;
const impact_ratio = [1.0 / 8, 1.0 / 8, 1.0 / 4, 1.0 / 2];

let chunks = this.chunks.slice(-frames_to_consider);
let normalized_chunk_size = 0;
for (let i = 0; i < frames_to_consider; i++)
  normalized_chunk_size += chunks[i].byteLength * impact_ratio[i];

const diff_bytes = normalized_chunk_size - frame_budget_bytes;
const diff_ratio = diff_bytes / frame_budget_bytes;

let qp_change = 0;
// Addressing overshoot more aggressively that undershoot
// Don't change QP too much when it's already low, because it
// changes chunk size to drastically.
if (diff_ratio > 0.6 && qp > 15) {
  qp_change = 3;
} else if (diff_ratio > 0.25 && qp > 5) {
  qp_change = 2;
} else if (diff_ratio > 0.04) {
  // Overshoot by more than 4%
  qp_change = 1;
} else if (diff_ratio < (qp < 10 ? -0.10 : -0.04)) {
  // Undershoot by more than 4% (or 10% if QP is already low)
  qp_change = -1;
}

new_qp = this.clamp_qp(qp + qp_change);
```

1. Compare **weighted** sum of last 4 encoded frames with the frame budget.
2. In case of overshoot increase QP by 1, 2 or 3 depending of the magnitude of overshoot. For small QP values always increase only by 1.
3. In case of undershoot decrease QP by 1

Discussion (End Time: 15:10)

-

Hardware Encode/Decode Error Handling (Bernard & Fippo)

Start Time: 15:10

End Time: 15:25

For Discussion Today

- WebRTC-Extensions
  - Issue 146: Exposing decode errors / SW fallback as an event
- WebCodecs
  - Issue 669: Custom Error Types

Issue 146: Exposing decode errors / SW fallback as an event

- In WebRTC, encode and decode errors can occur outside of SDP negotiation. Examples:
  - Failover from a hardware to software encoder
    - Due to resource availability
  - Failover from a hardware to software decoder
    - Due to a parsing error
    - Due to resource availability
  - Switch (without error) from HW to SW and vice versa
    - Due to quality (e.g. software offers better quality than hardware at low resolutions) and resolution change
- Caveat: hardware-only codecs
  - some H264 profiles on Windows, HEVC
  - No software fallback possible

Issue 146: Exposing decode errors / SW fallback as an event

- Is this actionable?
  - Application can notify the user to quit another application contending for hardware resources
  - Application can lower input resolution or frame rate to reduce encoder load
  - Application can signal the other side to reduce bitrate, framerate or the number of streams to reduce decoder load
  - Application can try to reset the encoder/decoder by deactivating and reactivating the transceiver
  - Application can log the input causing the error for further investigation

Issue 146: Exposing decode errors (cont'd)

- WebIDL: attribute EventHandler on(encode|decode)error;
  - On which object?
- Add encoder|decoder event handler to RTCPeerConnection
  - Use `mid` to identify which encoder/decoder this relates to
- Add event handler to RTCRtpSender
  - Use `rid` to identify which encoder this relates to
- Add event handler to RTCRtpReceiver

Issue 146: Exposing decode errors (cont'd)

- Proposal A: Reuse **`RTCError`** event
  - Need additional **`RTCErrorDetailType`** values. Examples:
    - **`Hardware-decoder-not-available`**
    - **`Hardware-decoder-parsing-error`**
  - Need to add attributes to **`RTCErrorInit`**. Examples:
    - simulcast layer identifier (rid)
    - timestamp

WebIDL

```
dictionary RTCErrorInit {
  required RTCErrorDetailType errorDetail;
  long sdpLineNumber;
  long sctpCauseCode;
  unsigned long receivedAlert;
  unsigned long sentAlert;
};
```

WebIDL

```
enum RTCErrorDetailType {
  "data-channel-failure",
  "dtls-failure",
  "fingerprint-failure",
  "sctp-failure",
  "sdp-syntax-error",
  "hardware-encoder-not-available",
  "hardware-encoder-error"
};
```

Issue 146: Exposing decode errors (cont'd)

- Proposal B: Custom event (like `RTCPeerConnectionIceErrorEvent`)

```
[Exposed=Window]
interface RTCRtpSenderErrorEvent : Event {
  constructor(DOMString type, RTCRtpSenderErrorEventInit eventInitDict);
  readonly attribute DOMString? rid;
  readonly attribute unsigned short errorCode;
  readonly attribute USVString errorText;
  readonly attribute long long timestamp;
 };

[Exposed=Window]
interface RTCRtpReceiverErrorEvent : Event {
  constructor(DOMString type, RTCRtpReceiverErrorEventInit eventInitDict);
  readonly attribute unsigned short errorCode;
  readonly attribute USVString errorText;
  readonly attribute long long timestamp;
```

Issue 669: Custom Error Types

- Discussed at May 30, 2023 MEDIA WG meeting. Minutes here.
- Recommendation:
  - `EncodingError`: problem with data provided to the encoder/decoder
  - `OperationError`: resource issue (hardware resources not available)
- Implementation progress?

Discussion (End Time: 15:25)

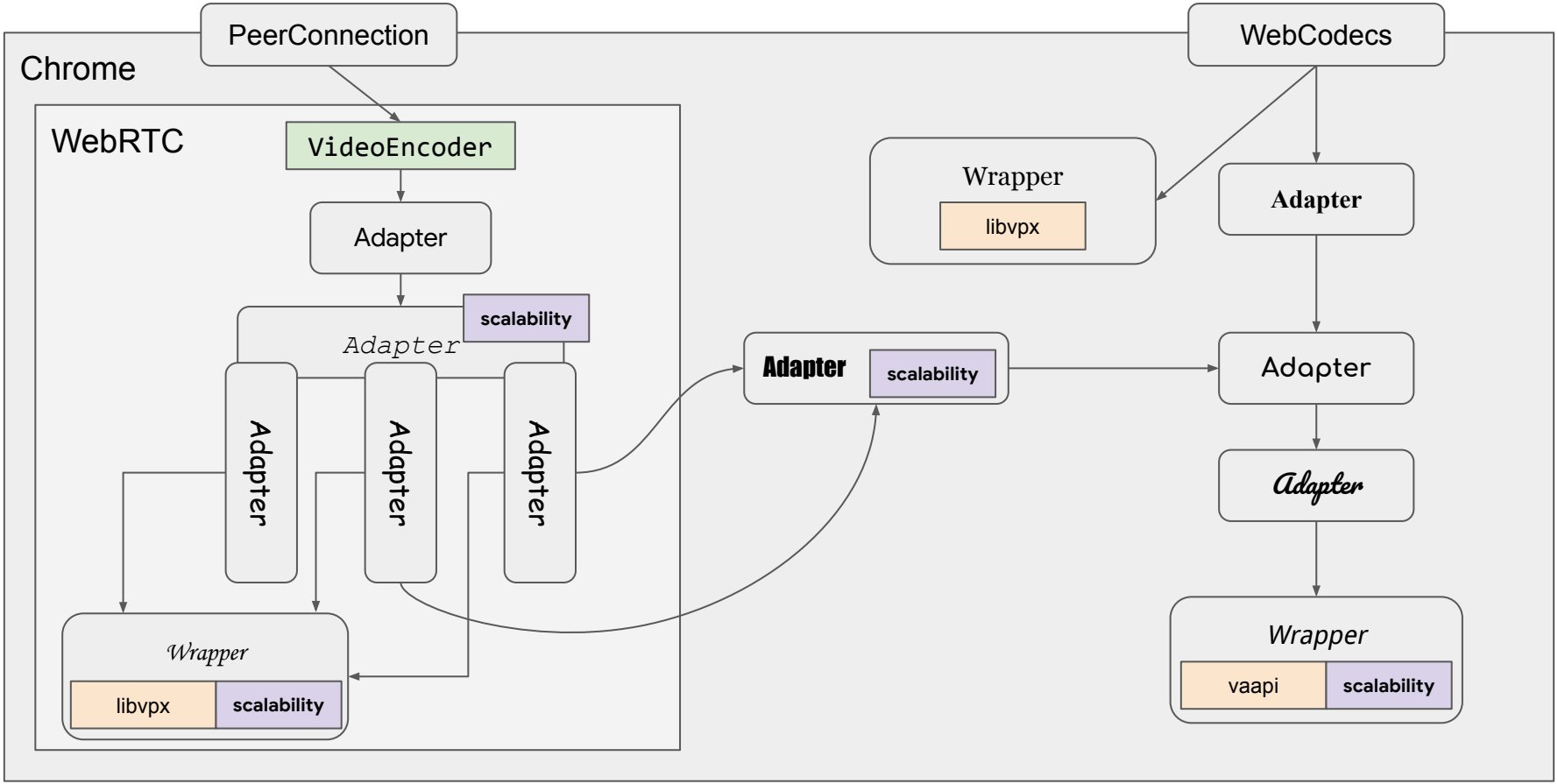
-

New Video Encoder API (Erik)

Start Time: 15:25

End Time: 16:25

Current State



Background

The Meet Video Quality team is planning a major overhaul of the internal WebRTC video encoder API. Our goals include:

- Make the encoders as low-level as possible
- Remove any RTP/Transport/PC related concepts
  - Only provide the minimal set of *tools* to implement scalability modes etc
- Be “just an encoder” - don’t tie it to RTC
- Be fully asynchronous

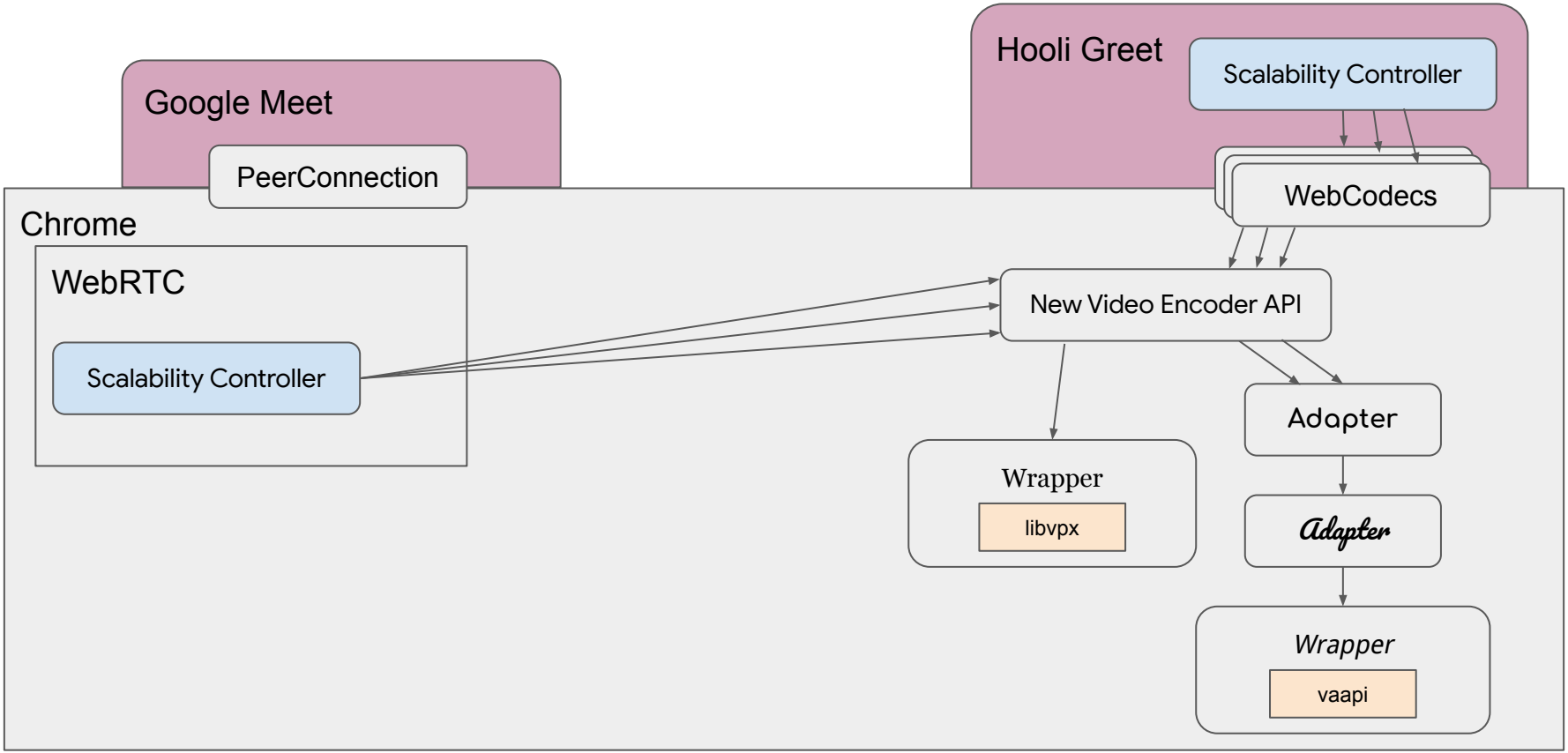
Background

Let’s use this opportunity to align WebCodecs and WebRTC!

tl;dr:

- Keep WebCodecs a low-level API
- Allows WebRTC & WebCodecs to share a lot of code
- Maybe even use WebCodecs as the encoders in WebRTC

Future Vision



Focus Areas

- Codec selection
- Flexible reference structures
- Minimize codec-specifics
- Rate control

Codec selection & prioritization

- In many cases the browser cannot automatically make the “best” choice.
- Acceptable quality/power draw/latency/compatibility is often dependent on both use-case and remote receivers.
- App needs to be able to *reason* about encoder behavior.

Codec selection & prioritization

Solution: Let the app be in full control!

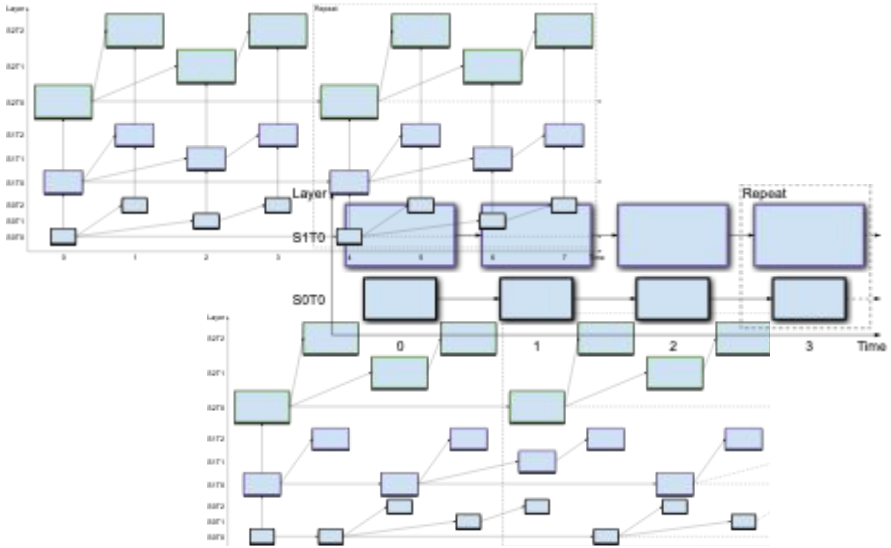
This implies that:

- One VideoEncoder instance contains one implementation.
- It’s possible to enumerate all available implementations.
- It’s possible to query the capabilities of each encoder.
- The app is fully responsible for error handling/fallbacks.

Flexible reference structures

Scalability modes - there's a lot of them!

Yet not enough…



Scalability Mode Identifier	Spatial Layers	Resolution Ratio	Temporal Layers	Inter-layer dependency	AV1 scalability_mode_idc
"L1T1"	1		1		N/A
"L1T2"	1		2		SCALABILITY_L1T2
"L1T3"	1		3		SCALABILITY_L1T3
"L2T1"	2	2:1	1	Yes	SCALABILITY_L2T1
"L2T2"	2	2:1	2	Yes	SCALABILITY_L2T2
"L2T3"	2	2:1	3	Yes	SCALABILITY_L2T3
"L3T1"	3	2:1	1	Yes	SCALABILITY_L3T1
"L3T2"	3	2:1	2	Yes	SCALABILITY_L3T2
"L3T3"	3	2:1	3	Yes	SCALABILITY_L3T3
"L2T1h"	2	1.5:1	1	Yes	SCALABILITY_L2T1h
"L2T2h"	2	1.5:1	2	Yes	SCALABILITY_L2T2h
"L2T3h"	2	1.5:1	3	Yes	SCALABILITY_L2T3h
"L3T1h"	3	1.5:1	1	Yes	
"L3T2h"	3	1.5:1	2	Yes	
"L3T3h"	3	1.5:1	3	Yes	
"S2T1"	2	2:1	1	No	SCALABILITY_S2T1
"S2T2"	2	2:1	2	No	SCALABILITY_S2T2
"S2T3"	2	2:1	3	No	SCALABILITY_S2T3
"S2T1h"	2	1.5:1	1	No	SCALABILITY_S2T1h
"S2T2h"	2	1.5:1	2	No	SCALABILITY_S2T2h
"S2T3h"	2	1.5:1	3	No	SCALABILITY_S2T3h
"S3T1"	3	2:1	1	No	SCALABILITY_S3T1
"S3T2"	3	2:1	2	No	SCALABILITY_S3T2
"S3T3"	3	2:1	3	No	SCALABILITY_S3T3
"S3T1h"	3	1.5:1	1	No	SCALABILITY_S3T1h
"S3T2h"	3	1.5:1	2	No	SCALABILITY_S3T2h
"S3T3h"	3	1.5:1	3	No	SCALABILITY_S3T3h
"L2T2_KEY"	2	2:1	2	Yes	SCALABILITY_L3T2_KEY
"L2T2_KEY_SHIFT"	2	2:1	2	Yes	SCALABILITY_L3T2_KEY_SHIFT
"L2T3_KEY"	2	2:1	3	Yes	SCALABILITY_L3T3_KEY
"L2T3_KEY_SHIFT"	2	2:1	3	Yes	SCALABILITY_L3T3_KEY_SHIFT
"L3T1_KEY"	3	2:1	1	Yes	
"L3T2_KEY"	3	2:1	2	Yes	SCALABILITY_L4T5_KEY
"L3T2_KEY_SHIFT"	3	2:1	2	Yes	SCALABILITY_L4T5_KEY_SHIFT
"L3T3_KEY"	3	2:1	3	Yes	SCALABILITY_L4T7_KEY
"L3T3_KEY_SHIFT"	3	2:1	3	Yes	SCALABILITY_L4T7_KEY_SHIFT

Flexible reference structures

Many structures are cumbersome or impossible to implement with scalability modes:

- Different variations on LTR
- Quality layers
- Out-of-order encoding (Bidirectional prediction)
- Dynamic framerate
- [insert your own novel strategy here]

Quickly becomes infeasible to implement this for every encoder.

Flexible reference structures

Solution: Let the app be in full control!

Model the encoder as a **set of reference buffers**.

Let the user specify encode **settings per output frame**:

- Resolution of the output frame
- Target size of the output (i.e. num bits)
- Which buffers may be referenced
- Which buffers to update

Flexible reference structures

Let the app be in full control!

Benefits include (but not limited to):

- A user can implement ***any*** **reference structure**
- No changes needed to codec implementations for your new structure
- Can be done in a **codec agnostic** way
- Minimal feedback needed

Rate Control

Many hardware accelerated codecs have poor rate control - at least where RTC is concerned.

Solution: Let the app be in full control!

Using **external rate control**, adapting bitrate use frame QP

Improvements and bug fixes can reside in the app - no need to wait for driver updates.

Draft API Proposal

For illustrative purposes (mostly)!

1: Encoder Capabilities

How we specify the characteristics of an encoder

Draft API - Capabilities

```
enum RateControlModes { kCQP, kCBR }
class BitrateControl {
  frameDropping: boolean;
  qpRange: [number, number];
  supportedModes: set<RateControlModes>;
}
```

```
class EncoderCapabilities {
  bitrateControl: BitrateControl;
  predictionConstraints: PredictionConstraints;
  inputConstraints: InputConstraints;
  encodingFormats: EncodingFormat[];
  performance: Performance;
}
```

Draft API - Capabilities

```
class PredictionConstraints {
  totalBuffers: number;
  maxReferences: number;

  maxTemporalLayers = 1;
  maxSpatialLayers = 1;

  scalingFactors: set<number> = [1];
  sharedBufferSpace = false;
}
```

```
class EncoderCapabilities {
  bitrateControl: BitrateControl;
  predictionConstraints: PredictionConstraints;
  inputConstraints: InputConstraints;
  encodingFormats: EncodingFormat[];
  performance: Performance;
}
```

Draft API - Capabilities

```
enum PixelFormat { kI420, kNv12,
kP016_LE, … }
class InputConstraints {
  min: Resolution;
  max: Resolution;
  pixelAlignment: number;
  inputFormats: PixelFormat[];
}
```

```
class EncoderCapabilities {
  bitrateControl: BitrateControl;
  predictionConstraints: PredictionConstraints;
  inputConstraints: InputConstraints;
  encodingFormats: EncodingFormat[];
  performance: Performance;
}
```

Draft API - Capabilities

```
enum SubSampling { k420, k422, k444 }
class EncodingFormat {
  subSampling: SubSampling;
  bitdepth: number;
}
```

```
class EncoderCapabilities {
  bitrateControl: BitrateControl;
  predictionConstraints: PredictionConstraints;
  inputConstraints: InputConstraints;
  encodingFormats: EncodingFormat[];
  performance: Performance;
}
```

Draft API - Capabilities

```
class Performance {
  maxEncodedPixelsPerSeconds?: number;
  minMaxEffortLevel: [number, number];
}
```

```
class EncoderCapabilities {
  bitrateControl: BitrateControl;
  predictionConstraints: PredictionConstraints;
  inputConstraints: InputConstraints;
  encodingFormats: EncodingFormat[];
  performance: Performance;
}
```

2: Encoder Creation

- Get list of encoder factories
- Each factory creates instances of a **single implementation**

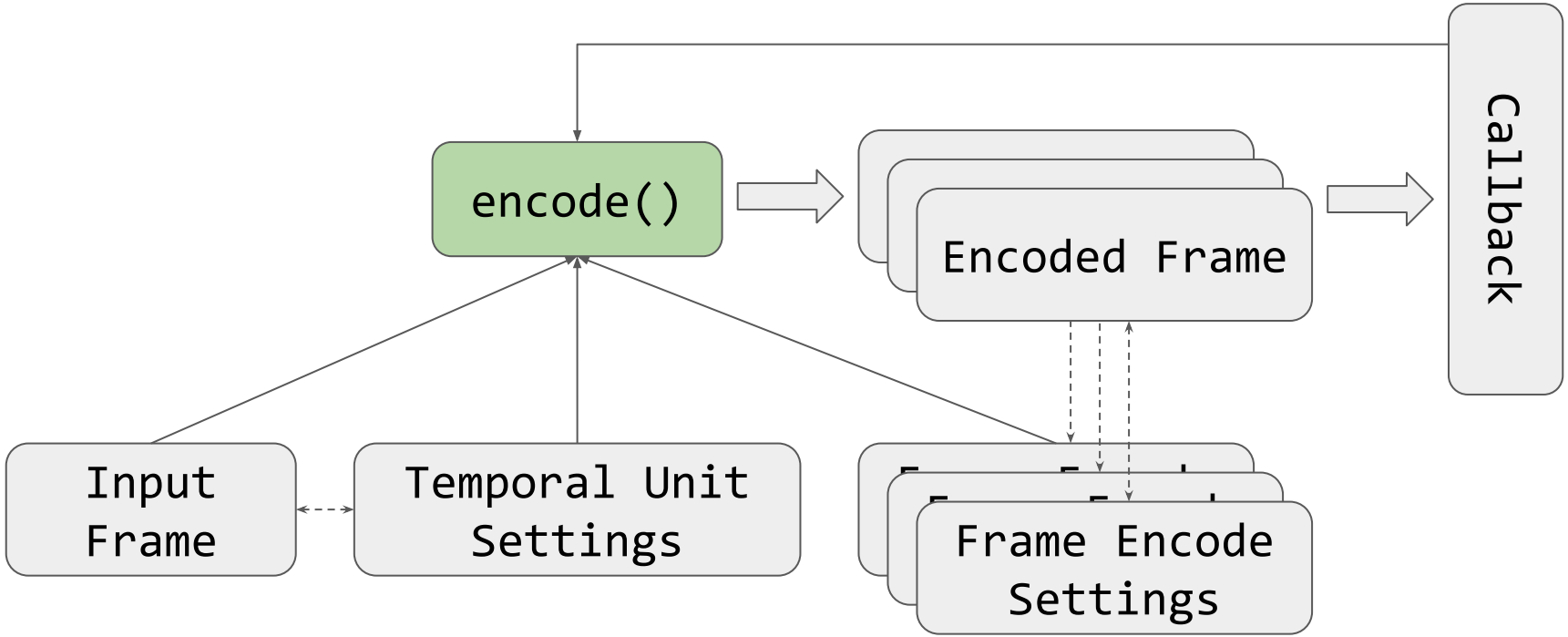
Draft API - Create encoder

```
function enumerateVideoEncoders(): VideoEncoderFactory[]

interface VideoEncoderFactory {
  getCapabilities(): EncoderCapabilities;
  getImplementationName(): string;
  getCodecName(): string;
  getCodecSpecifics(): map<string, string>;

  createVideoEncoder(settings: EncoderSettings): VideoEncoder;
}
```

Draft API - Create encoder

```
class EncoderSettings {
  maxNumberOfThreads: number;
  maxEncodeDimensions: Resolution;
  encodingFormat: EncodingFormat;
  rcMode: RateControlModes;
  codecSpecifics: map<string, string>;
}
```

3: Encoding Frames



Draft API - Encode frames

```
enum ContentHint { kMotion, kDetail }
enum FrameDroppingMode { kOff, kAnyLayer, kAllLayers }
class TemporalUnitSettings {
  contentHint: ContentHint;
  effortLevel: number;
  frameDroppingMode: FrameDroppingMode;
}
```

```
interface VideoEncoder {
  encode(
    inputFrame: VideoFrame,
    settings: TemporalUnitSettings,
    frameEncodeSettings: LayerSettings[],
    callback: EncodeResultCallback): bool;
}
```

Draft API - Encode frames

```
class CbrParams {
  durationUs: number;
  dataRateBps: number;
}
class CqpParams {
  targetQp: number;
}

class LayerSettings {
  rcOption: CbrParams|CqpParams;
  ...
}
```

```
interface VideoEncoder {
  encode(
    inputFrame: VideoFrame,
    settings: TemporalUnitSettings,
    frameEncodeSettings: LayerSettings[],
    callback: EncodeResultCallback): bool;
}
```

Draft API - Encode frames

```
class LayerSettings {
  rcOption: CbrParams|CqpParams;
  forceKeyframe: boolean;
  temporalId: number;
  spatialId: number;
  resolution: Resolution;
  referenceBuffers: set<number>;
  updateBuffers: set<number>;
}
```

```
interface VideoEncoder {
  encode(
    inputFrame: VideoFrame,
    settings: TemporalUnitSettings,
    frameEncodeSettings: LayerSettings[],
    callback: EncodeResultCallback): bool;
}
```

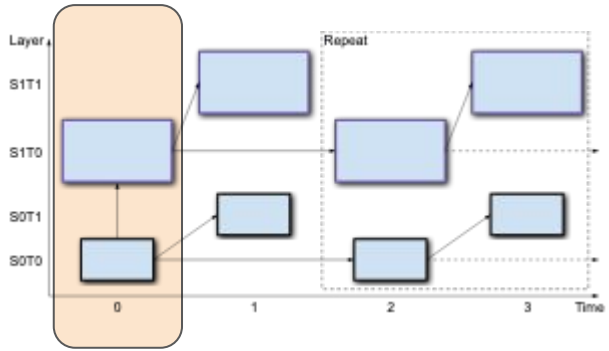
Draft API - Encode frames

```
enum DropReason { kDropped, kError }
class DroppedFrame {
  reason: DropReason;
  spatialId: number;
}

class EncodedData {
  bitstreamData: EncodedVideoChunk;
  isKeyframe: boolean;
  spatialId: number;
  referencedBuffers: set<number>;
}
type EncodeResultCallback = (result: EncodedData|DroppedFrame) => void;
```

```
interface VideoEncoder {
  encode(
    inputFrame: VideoFrame,
    settings: TemporalUnitSettings,
    frameEncodeSettings: LayerSettings[],
    callback: EncodeResultCallback): bool;
}
```

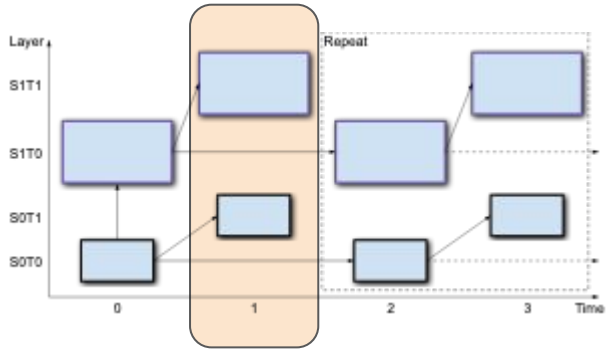
Example: Implementing L2T2_KEY

```
encoder.encode(
  inputFrame, encoderSettings,
  frameEncodeSettings: [{
    forceKeyframe: true,
    temporalId: 0,
    spatialId: 0,
    resolution: {320, 180},
    referenceBuffers: [],
    updateBuffers: []  // Implicitly all.
  }, {
    forceKeyframe: false,
    temporalId: 0,
    spatialId: 1,
    resolution: {640, 360},
    referenceBuffers: [0],
    updateBuffers: [1]
  }],
  callback);
```



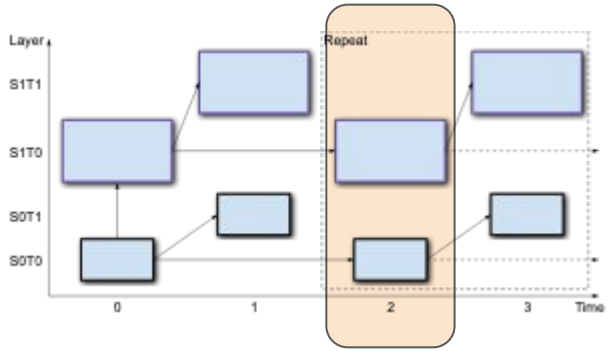
Example: Implementing L2T2_KEY

```
encoder.encode(
  inputFrame, encoderSettings,
  frameEncodeSettings: [{
    forceKeyframe: false,
    temporalId: 1,
    spatialId: 0,
    resolution: {320, 180},
    referenceBuffers: [0],
    updateBuffers: []
  }, {
    forceKeyframe: false,
    temporalId: 1,
    spatialId: 1,
    resolution: {640, 360},
    referenceBuffers: [1],
    updateBuffers: []
  }],
  callback);
```



Example: Implementing L2T2_KEY

```
encoder.encode(
  inputFrame, encoderSettings,
  frameEncodeSettings: [{
    forceKeyframe: false,
    temporalId: 0,
    spatialId: 0,
    resolution: {320, 180},
    referenceBuffers: [0],
    updateBuffers: [0]
  }, {
    forceKeyframe: false,
    temporalId: 0,
    spatialId: 1,
    resolution: {640, 360},
    referenceBuffers: [1],
    updateBuffers: [1]
  }],
  callback);
```



What do you mean, what “fingerprinting”?

Discussion (End Time: 16:25)

-

Wrapup and Next Steps

Start Time: 16:25

End Time: 16:30

Next Steps

- Content goes here

Thank you

Special thanks to:

WG Participants, Editors & Chairs