GstWebRTC / WebKit state of the union

Philippe Normand & Carlos Bentzen

GStreamer conference 2024

igalia

Outline

- Intro
- Pipeline changes between 2.42 and 2.46
- What’s new?
- Practical use-cases
- On-going work
- Plans

Intro

WebKit

- FOSS Web engine, maintained by Apple, Igalia, Sony
- WebView API, allowing to build Web browsers and many other products, including:
  - Set-top-box UIs
  - News readers
  - Containerized web-apps (Tangram)
- Platform-specific ports:
  - Apple products
  - Sony Playstation
  - Linux Desktop (GTK port)
  - Linux Embedded (WPE port)

WebRTC

- Realtime Communication for the Web
- Use-cases:
  - Video chatting (Google meet, etc)
  - Cloud gaming
  - Online auctions, sport events betting
  - Broadcasting
  - Beyond the web browser, with native apps

WebRTC in WebKit

- `getUserMedia()`: Access to camera/microphone (with PipeWire or video4linux) 🚀
- `getDisplayMedia()`: Access to screen contents (PipeWire, using screencast desktop portal) 🚀
- `MediaStream`: Playback of streams, integration with `<video>`,canvas, webgl, webaudio,... 🚀
- `RTCPeerConnection`: P2P connection (with `webrtcbin`) 🚧

Features currently not enabled by default, due to active development. They are enabled in developer builds and WebKit post-commit bots.

igalia

WebRTC WebKit backends

- LibWebRTC:
  - Basic GStreamer integrations (video decoding)
  - Not shipped in releases
- GstWebRTC:
  - Relying on `webrtcbin` and `libgstwebrtc`
  - Hopefully will ship in releases ;)

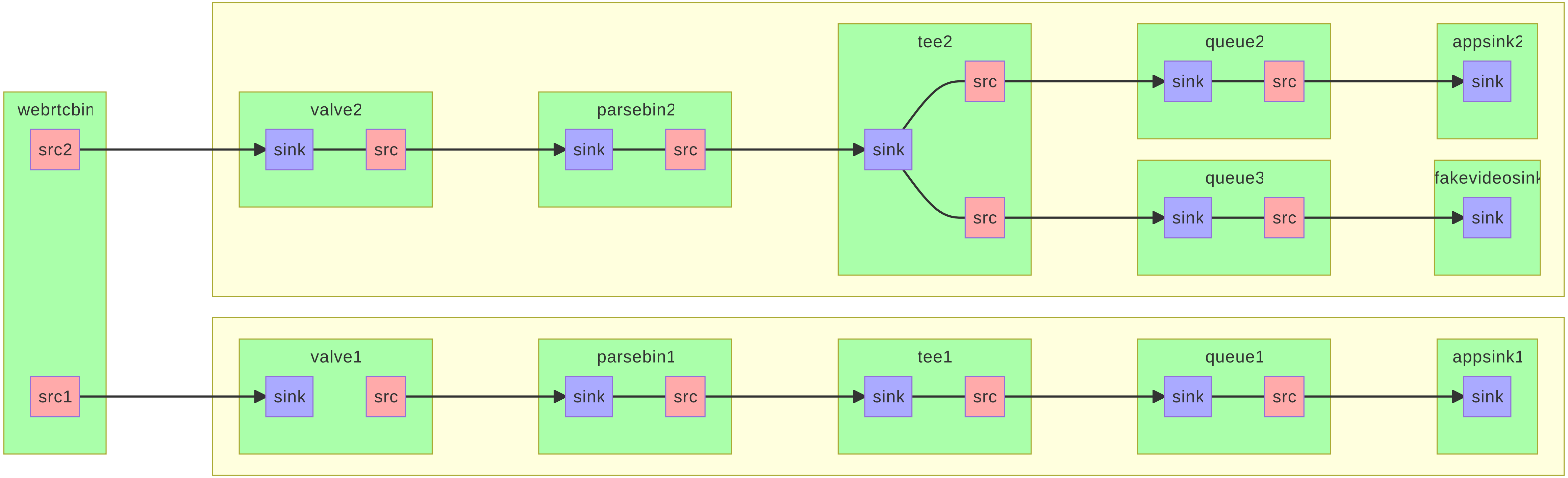
Pipeline changes between 2.42 and 2.46

igalia

Recap: the pipelines involved

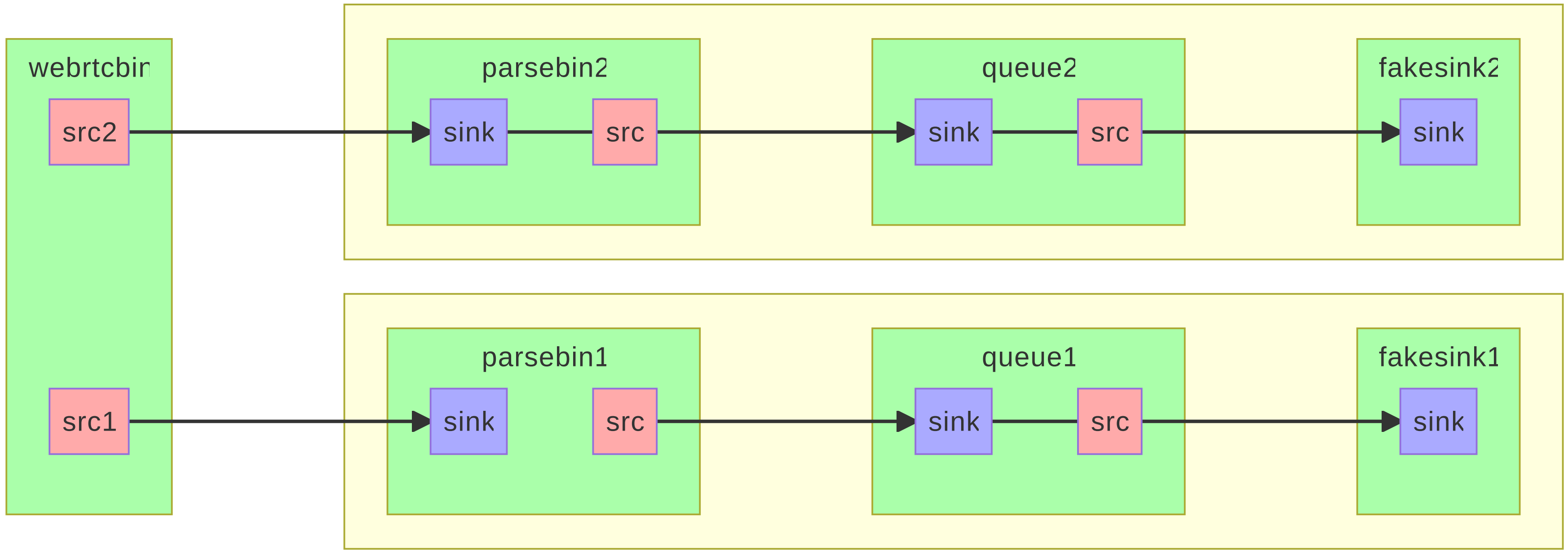
- WebRTC pipeline: where `webrtcbin` lives
- Capture pipeline: where capture elements like `pipewiresrc` are
- Playback pipeline: where `playbin3` lives

Receiving side in 2.42



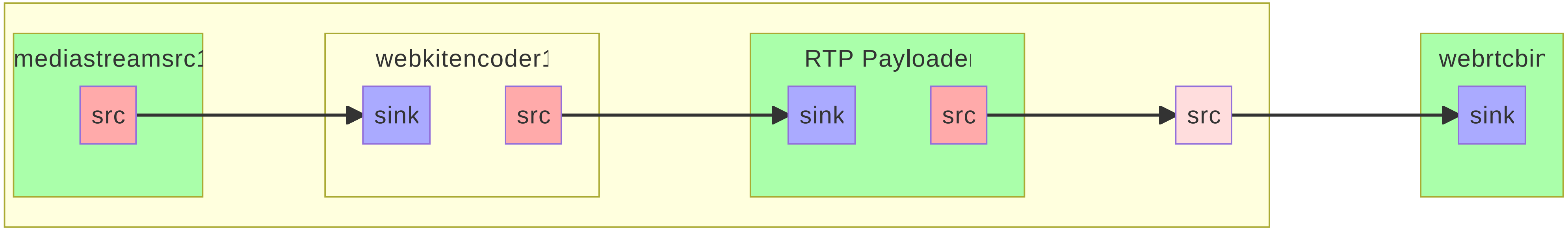
- valve element used for muted / inactive tracks
- tee -> (fakevideosink + appsink) for stats in tests

Receiving side in 2.46



- No more dynamic pipeline changes with tee elements
- Buffers forwarded from fakesink `handoff` signal to WebKit observers

Sender side



- New video encoders: av1enc, vaav1enc

`replaceTrack()` simplification (1/2)

In JavaScript: `peerConnection.replaceTrack(oldTrack, newTrack);`

In 2.42:

- the `mediastreamsrc` element associated with `oldTrack` was removed from the pipeline
- A new `mediastreamsrc` element was added dynamically, observing `newTrack`

`replaceTrack()` simplification (2/2)

In 2.46:

- Existing `mediastreamsrc` stops `oldTrack` observation
- Starts observing `newTrack`
- No issue with caps handling, managed by `appsrc` within `mediastreamsrc`.

What’s new?

Track events

```
let onTrackFired = false
const pc = new RTCPeerConnection()
pc.ontrack = e => {onTrackFired = true}
await pc.setRemoteDescription(remoteOffer)
console.assert(onTrackFired)
```

- Earlier creation of transceivers in webrtcbin to be more spec compliant
- To be released in GStreamer 1.26

igalia

Legacy Offer options

- `pc.createOffer({offerToReceiveAudio: true, offerToReceiveVideo: true})`
- Still supported in Chrome browsers, needed for improved interop
- Supported in WebKit, behind a runtime flag
- Implementation on the cross-platform layer, by adding a `recvonly` transceiver, depending on the options given.

RTP2 payloaders

- Re-implementations of the (de)payloders from -good, in Rust, shipping in gst-plugins-rs
- Same public interface (properties, signals) as the C versions
- In WebKit:
  - rank-based payloader selection in order to generate offer/answers
  - for incoming RTP streams, parsebin is used, will also select depayloader based on rank

Practical use-cases

Amazon Luna

- Already demoed at GStreamer Conference 2023
- Game streaming, gamepad events sent using DataChannel
- Optional support for live WebCam/Mic overlay to Twitch (WIP)
- In 2024 we kept up with the web app changes and maintenance
- **Demo**

Zoo, Industrial modeling application

- It streams CAD model rendered server-side with WebRTC
- DataChannel for sending mouse events
- **Demo**

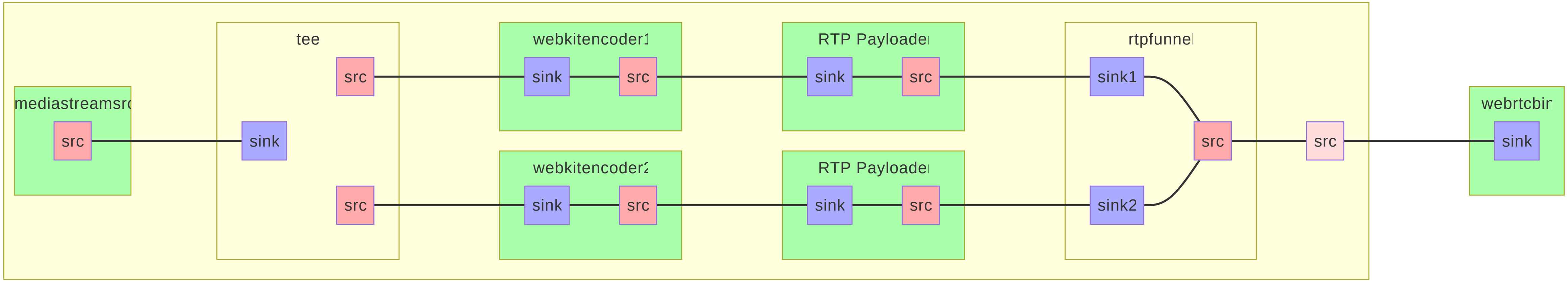
🚧 On-going work 🚧

Simulcast (1/4)

- Same captured stream, encoded multiple times at different bitrates and framerates
- Widely used by most SFUs in order to cope with varying receivers network conditions

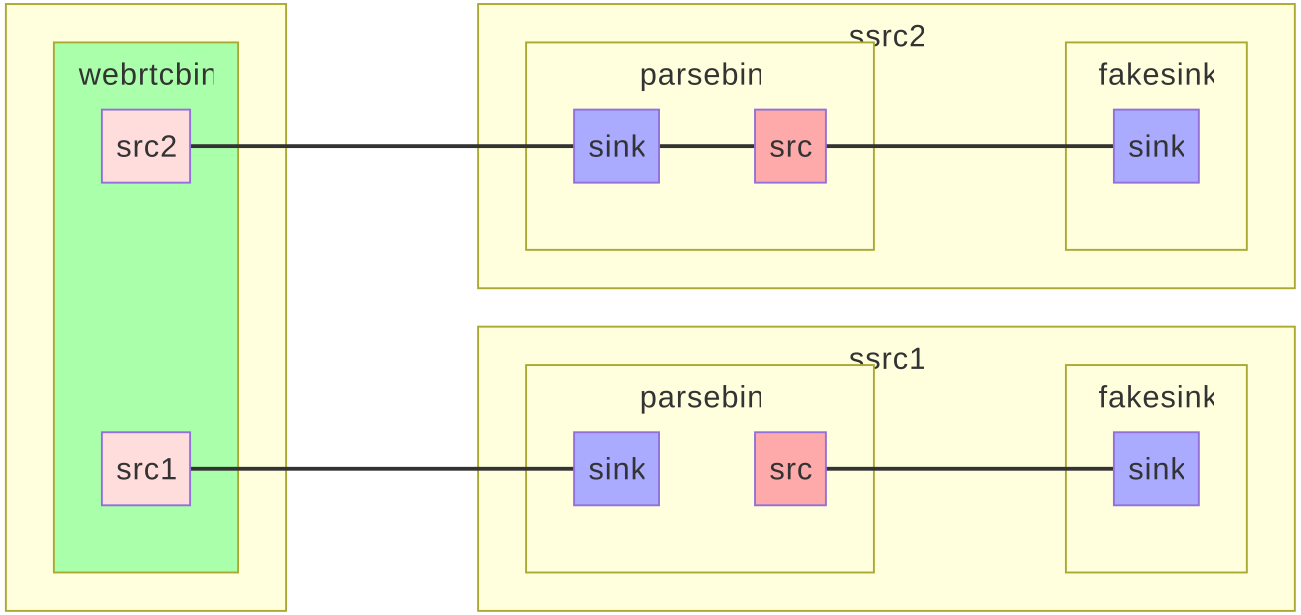
```
pc.addTransceiver(stream.getVideoTracks()[0], {
    direction: "sendonly",
    sendEncodings: [
      { rid: "h", maxBitrate: 1200 * 1024 },
      { rid: "m", maxBitrate:  600 * 1024, scaleResolutionDownBy: 2 },
      { rid: "l", maxBitrate:  300 * 1024, scaleResolutionDownBy: 4 }
    ]
});
```

Simulcast - sender side (2/4)



`rtpfunnel` aggregates encoded/payloaded streams and sends the merged stream to `webrtcbin`

Simulcast - receiver side (3/4)



`webrtcbin` exposes one src pad per simulcast stream / `ssrc`

Simulcast, on-going challenges (4/4)

- Sender side: Ensuring `mid` correctly written to RTP headers (FEC encoder rewrites those).
- Receiver side: Have `mid` set in pad caps (plumbing needed between ptdemux, rtpbin, webrtcbin).
- Shipped WebKit feature likely to depend on GStreamer >= 1.28, we will need to properly gate this at runtime.

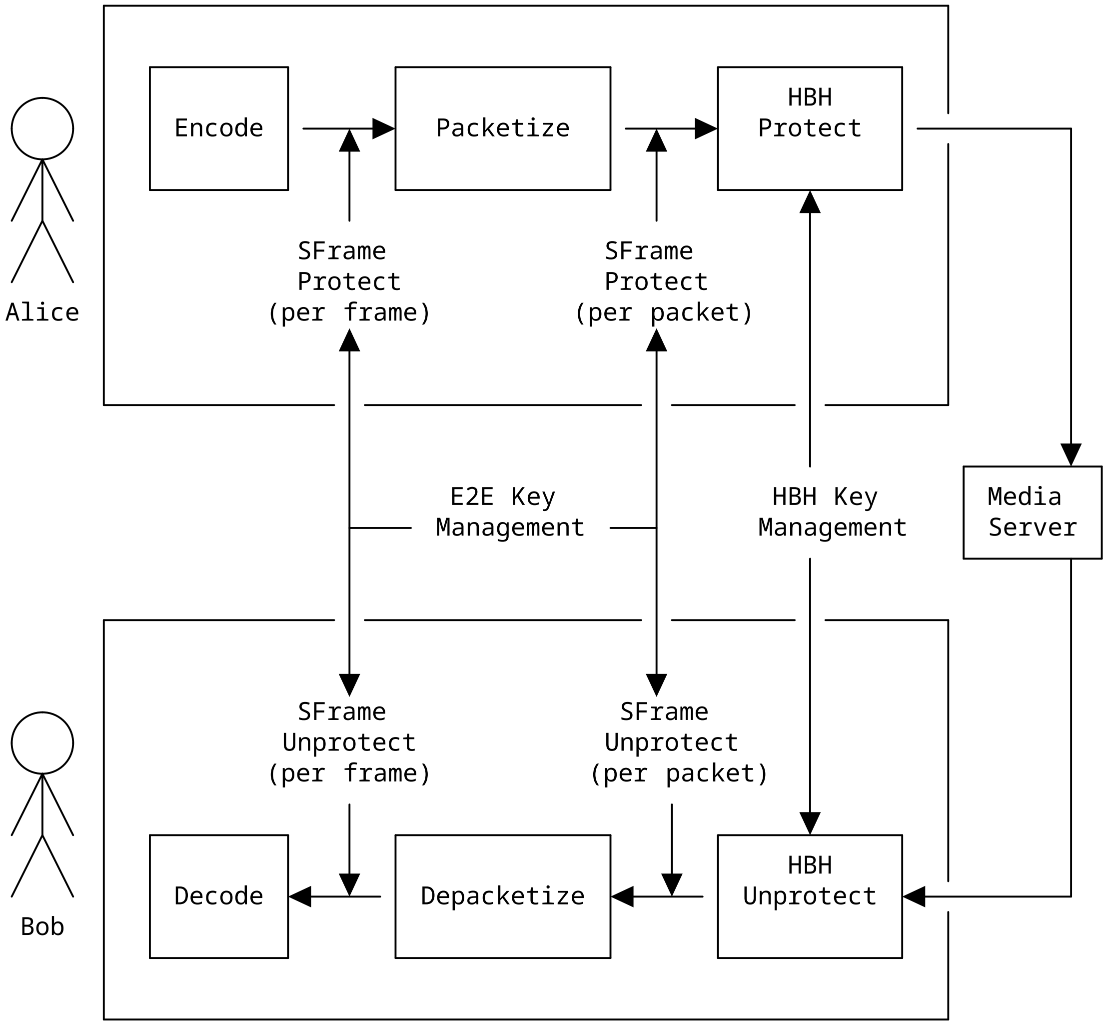
SFrame: the basics (1/5)

- Proposed standard: Lightweight Authenticated Encryption for Real-Time Media (RFC 9605)
- Combined with **WebRTC encoded transforms**

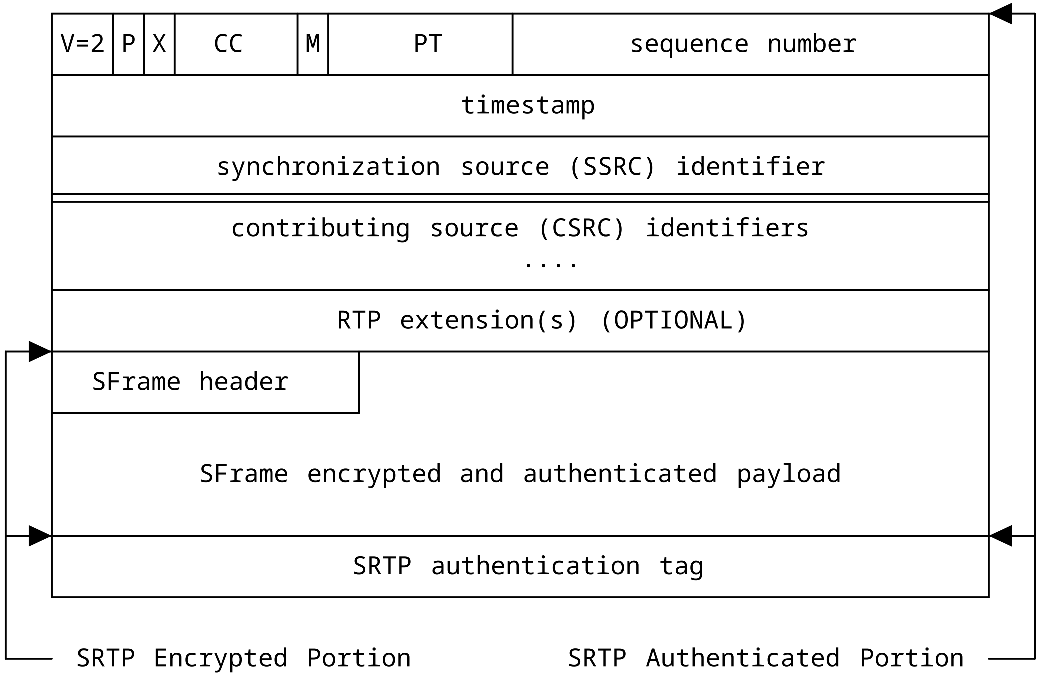
```
const key = await crypto.subtle.importKey("raw", new Uint8Array([...]), "HKDF", false, ["deriveBits", "deriveKey"]);
let transform = new SFrameTransform({compatibilityMode: "H264"});
transform.setEncryptionKey(key);
let sender = pc.addTrack(localStream.getTracks()[0], localStream);
sender.transform = transform;
```

Receiver has similar code, executed in `ontrack` event handler.

SFrame: workflow for E2EE (2/5)



SFrame: RTP encapsulation (3/5)



SFrame: Integration with WebKit's GStreamer backend (4/5)

- Encoded Transforms mandate full frame encryption
- So we would need custom “generic” RTP payloader and depayloader

SFrame, on-going challenges (5/5)

- ✅ We have a working OpenSSL backend for SFrame cypher handling
- 🚧 Integrating the callback-based encryption from WebKit to a new RTP (de)payloader remains an open topic
- 🚧 RTP payloading requires media metadata as well, still to be hooked from WebKit to GStreamer

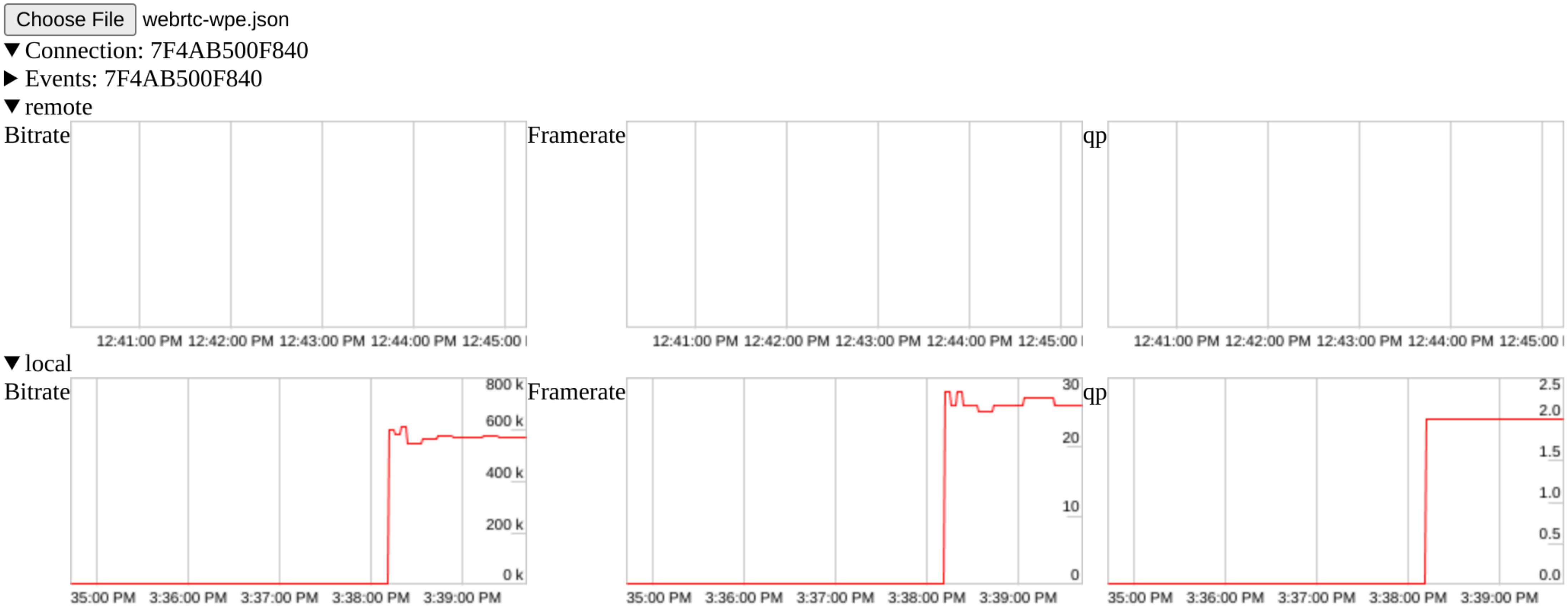
Devtools (1/3)

- WebRTC events and stats gathering for live graphing and post-mortem analysis
- Support for LibWebRTC and GstWebRTC WebKit builds
- JSON stream emitted by WebKit's PeerConnection backends, including timestamped events, example:


```
{"peer-connection":"7F1C6E013520","timestamp":1725960727633365.8,"type":"event","event":{"message":"PeerConnection creat
{"peer-connection":"7F1C6E013680","timestamp":1725960735855362.8,"type":"stats","event":{"type":"inbound-rtp","id":"rtp-
```


- Basic web frontend able to read such JSON file and render graphs

Devtools (2/3)



Graphs showing basic stats about inbound and outbound RTP streams

Devtools (3/3)

- Backend submitted for review (needs additional iterations)
- Frontend:
  - Standalone proof-of-concept
  - Integration in WebKit pending (either in WebInspector or a custom `webkit://webrtc-internals` handler)
- Hopefully shipping in 2.48 🤞

Plans regarding Sandboxing

Camera portal support

- The current capture device pipeline runs in WebProcess (BAD!)
- Ideally the WebProcess should be sandboxed, hence no direct access to capture devices
- Currently we allow-list v4l devices in the sandbox
- Plan (2025): Integration with the desktop Camera portal, giving us access to PipeWire nodes
- WPE on Embedded platforms will still need v4l devices allow-listing

librice integration

- The current streaming pipeline runs in WebProcess (BAD!)
- Ideally the network usage should be restricted to the NetworkProcess
- libgstwebrtc now supports custom ICE implementations
- librice provides a Sans-IO implementation for ICE handling
- Plan (2025): Implement a librice-based ICE backend in the WebProcess, handling IPC I/O with NetworkProcess

Thanks!

Any question?