Insertable Media Processing

An API proposal for RTCPeerConnection

Harald Alvestrand

TPAC 2019

Requirements

- Allow user-specified processing of media in Web applications
    - Typically, WASM is quoted as “the tool to use”
- Allow this at reasonable performance
- Allow this without being interrupted by other Web page activities

The present solution

- Pump the stream to a <video> element
- Capture the video using a <canvas> element
- Either paint on the canvas, or extract the bits using ImageData
- Recompose the video using “capture from canvas”

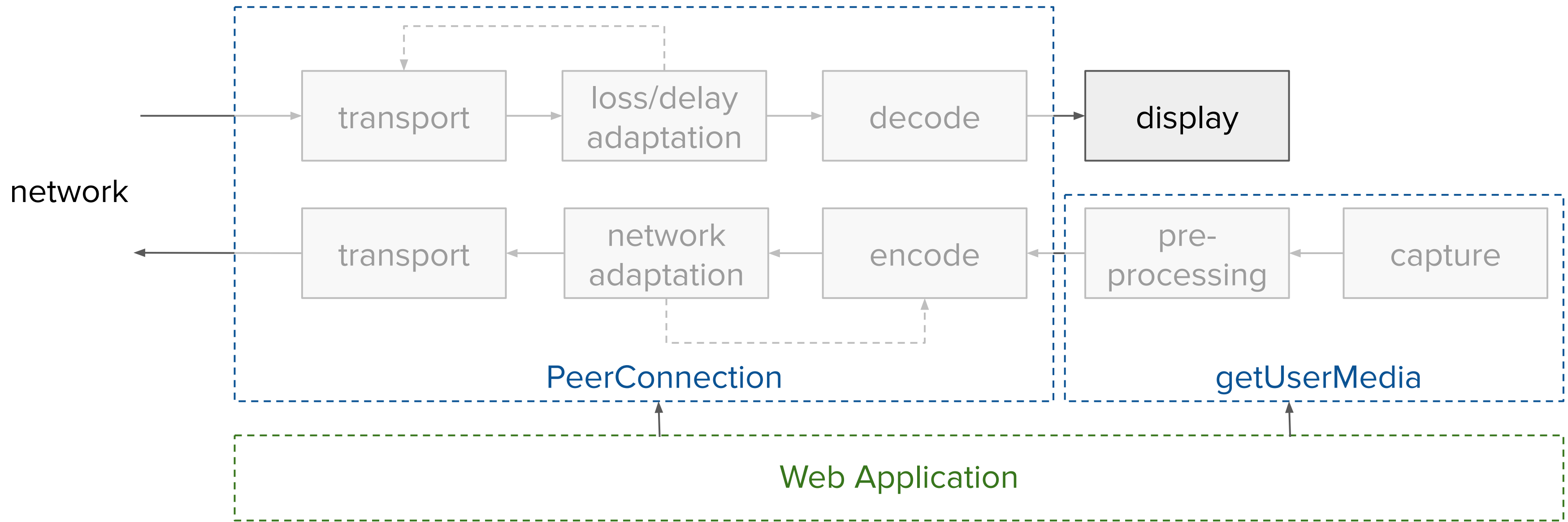
Problems:

- Not very efficient
- Somewhat painful to code
- Doesn’t allow for optimizations by the browser

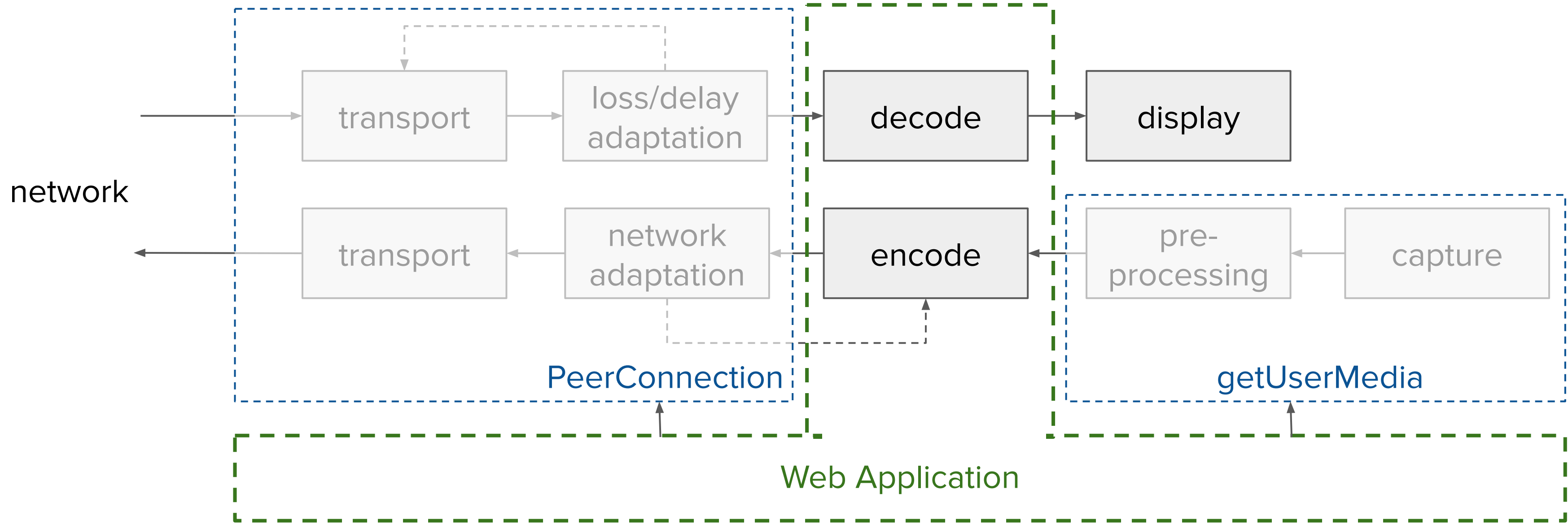
Nice-to-have requirements

- Minimally disruptive to existing applications
  - Don’t require new protocols, new APIs
  - Reuse libraries and code that works with WebRTC 1.0
- Don’t block the future
  - We believe the future is composable components
  - Expose components that will exist in future models

RTC media flow in WebRTC 1.0



Insertable Streams encode/decode flow



Wrapping the encoder (or decoder)

An encoder (from the WebCodec proposal) consists of:

- An input stream (WHATWG stream) of unencoded frames + control messages
- An output stream of encoded frames
- A processing element that does the actual encoding

We can use the WHATWG piping operations to fit processing before or after the encoder

This requires access to the encoder at the time it's created (reattachment has issues)

Code Example

Example code for a processing step running in a worker

```
pc = new RTCPeerConnection({
    encoderFactory: (encoder) => {
        var munger = new TransformStream({ transformer: munge });
        output = encoder.readable.pipeThrough(munger.writable);
        worker.postMessage([‘munge this’, munger], [munger]);
        return { readable: output, writable: encoder.writable };
    }
  })});
```

For the step-by-step version, see the explainer.

Pre-solved Stuff when picking this model

Transferable Streams allows us to send the stream endpoints to a worker for processing, without having to define anything new. No need to give Worker access to PeerConnection or getUserMedia!

Memory handling is folded into the WebCodec project, and can be solved there.

The Streams standard allows optimizations to be taken wherever they can't be observed - which means that two internal processing steps can be linked without visiting JS space.