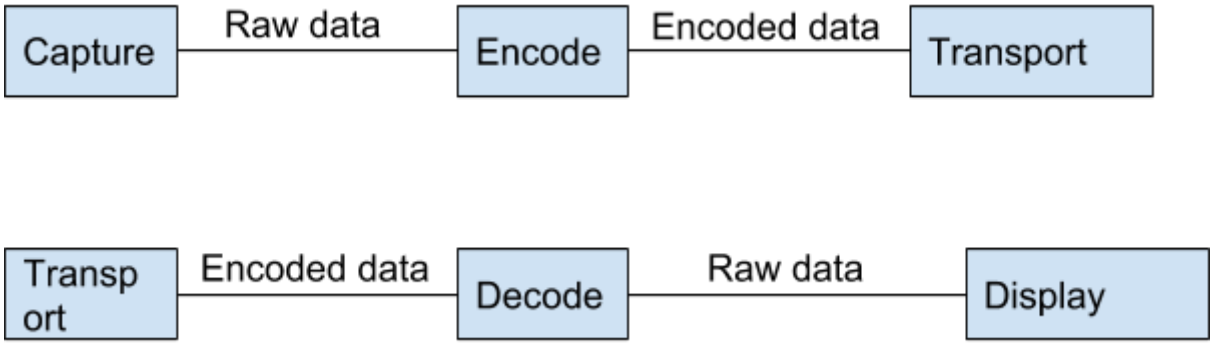
Certain things are hard to do in the present WebRTC / MediaStreamTrack API.
In particular, anything involving manipulation of raw data involves convoluted interfaces that impose burdens of format conversion and/or buffer copying on the user.

This document sketches the use cases that can be made possible if this access is made a lot easier and with lower overhead.

For reference, a model of the encoding / decoding pipeline in the communications use case:



When doing other types of processing, the pipeline stages may be connected elsewhere; for instance, when saving to file (MediaRecorder), the “Encode” step links to a “Storage” step, not “Transport”.

The “Decode” process will include alignment of media timing with real time (NetEq / jitter buffer); the process from raw data to display will happen “as fast as possible”.

Raw Image Use Cases

This set of use cases involves the manipulation of video after it comes from the camera, but before it goes out for transmission, or vice versa.
Examples of apps that consume raw data from a camera or other source, producing raw data that goes out for processing:

- Funny hats
- Background removal
- In-browser compositing (merge video streams)

Needed APIs:
- Get raw frames from input device or path
- Insert (processed) raw frames into output device or path

Non-Standard Encoders

This set of tools can be useful for either special types of operations (like detecting face movement and sending only those for backprojection on a model head rather than sending

the picture of the face) or for testing out experimental codecs without involving browser changes (such as novel SVC or simulcast strategies).

Needed APIs, send side:

- Get raw frames from input device
- Insert encoded frames on output transmission channel
- Manipulate transmission setup so that normal encoder resources are not needed

Needed APIs, receive side:

- Signalling access so that one knows what codec has been agreed for use
- Get encoded frames from the input transmission channel
- Insert raw (decoded) frames into output device or path

Pre/post-transmission processing - Bring Your Own Encryption

This is the inverse of the situation above: One has a video stream and wishes to encode it into a known codec, but process the data further in some way before sending it.
The example in the title is one use case.
The same APIs will also allow the usage of different transmission media (media over the data channel, or media over protobufs over QUIC streams, for instance).

Needed APIs, encode:

- Codec configuration - the stuff that usually happens at offer/answer time
- Getting the encoded frames from the “output” channel
- Inserting the processed encoded frames into the real “output” channel
- Reaction to congestion information from the output channel
- Feeding congestion signals into the encoder

Needed APIs, decode:

- Codec configuration information
- Getting the encoded frames from the input transport
- Inserting the processed encoded frames into the input decoding process

The same APIs are needed for other functions, such as:

- ML-NetEq: Jitter buffer control in other ways than the built-in browser
    - This also needs the ability to turn off the built-in jitter buffer, and therefore makes this API have the same timing requirements as dealing with raw data
- ML-FEC: Application-defined strategies for recovering from lost packets.
- Alternative transmission: Using something other than browser’s built-in realtime transport (currently SRTP) to move the media data