RTCRtpSender/Receiver

Justin Uberti, May 2014

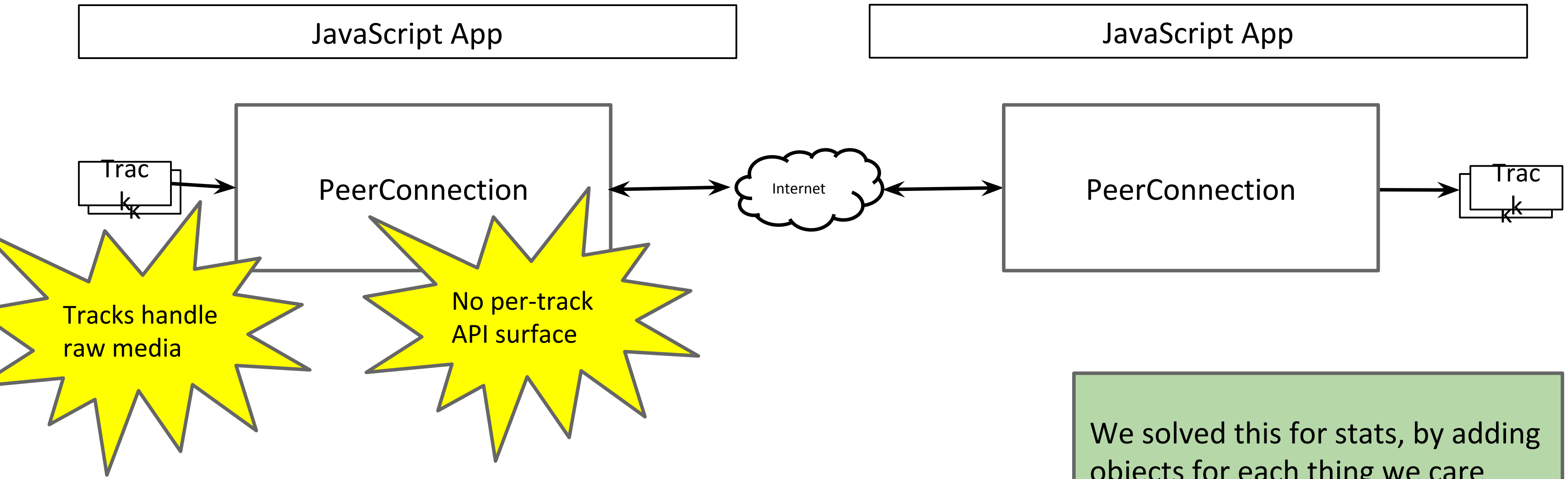
The Basics

(things that should be uncontroversial)

Review: Problem Statement

- Need a way to tweak params on individual tracks sent over the wire, e.g.
  - **Bitrate**
  - **Direction** (sendonly/recvonly etc)
- Existing control surfaces insufficient
  - **createOffer params** - not per-track
  - **AddStream params** - not modifiable post-add
  - **MST constraints** - affects raw media, not encoding

Core Issue: Insufficient Object Model

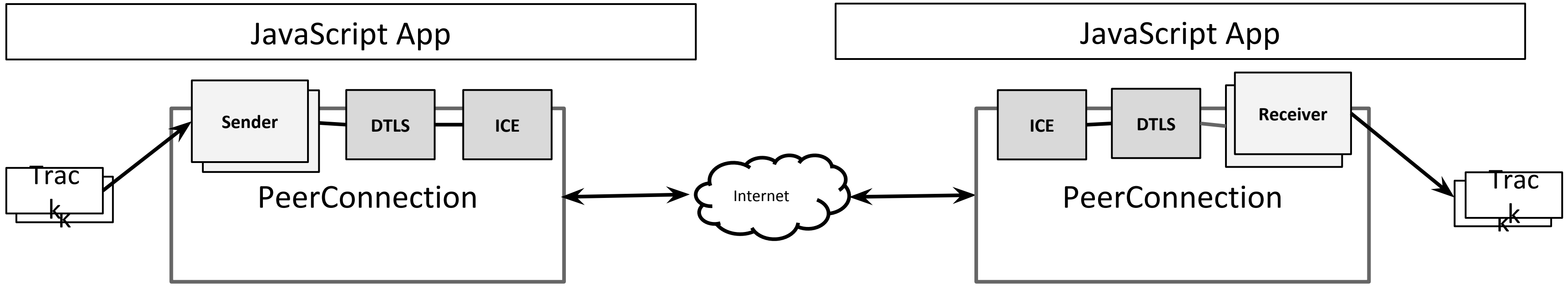


We solved this for stats, by adding objects for each thing we care about, but these objects are hidden inside PeerConnection

Solution

- Expose the objects that represent the things that apps want to change
  - **RTCRtpSender** (converts raw media into packets)
  - **RTCRtpReceiver** (converts packets into raw media)
  - Both are 1:1 with MediaStreamTracks
    - A RTCRtpSender encodes a single track
    - A RTCRtpReceiver produces a single track
    - However, there may be **multiple encodings**

Solution Diagram



Applications now have an API surface with the right multiplicity to do per-track operations

Obtaining RTCRtpSender/Receiver

- **RTCRtpSender** created/returned when you add a local track:
  - `sender = pc.addTrack(mst);`
- **RTCRtpReceiver** vended when a remote track is added
  - `function onaddtrack(e) { receiver = e.receiver; }`
- Trivially gettable from PC
  - `pc.getSenders(), pc.getReceivers()` each return sequences

- Makes for clear 1:1 relationship with tracks, but requires us to replace the existing AddStream/onaddstream etc APIs with track-specific versions

Streams -> Tracks

- Most operations are simple replacements:
    - **removeStream** -> **removeTrack**
    - **getLocalStreams** -> **getSenders**
    - **getRemoteStreams** -> **getReceivers**
    - **onaddstream** -> **onaddtrack**
- And, trivially polyfillable for backwards compat:
    -
      ```
      function removeStream(s) {
        for (var i = 0; i < s.getAudioTracks().length; ++i)
          this.removeTrack(s.getAudioTracks()[i]);
        for (i = 0; i < s.getVideoTracks().length; ++i)
          this.removeTrack(s.getVideoTracks()[i]);
      }
      ```

Special case: addTrack

- A track can be part of multiple streams. What should it communicate to the other side?
    - **Nothing**: app should put together its own streams
        - Pro: Simple
        - Con: Change in app behavior (now get separate streams for a/v)
    - **Everything**: all stream associations should be communicated
        - Pro: Sender actions mirrored at receiver
        - Con: Complex. Adding a track to a new stream will require an offer/answer exchange, and could change receiver experience.
    - **Minimum**: a single stream association.
        - Pro: No behavior change for current apps (get one a/v stream)
        - Con: Multi-stream sync requires explicit handling by app

addTrack proposal

- Suggestion: take the minimal approach

  -
    ```
    RTCRtpSender addTrack(MediaStreamTrack track,
                          MediaStream stream = null);
    ```

    ```
    pc.addTrack(camStream.getAudioTracks()[0], camStream);
    pc.addTrack(camStream.getVideoTracks()[0], camStream);
    pc.addTrack(desktopStream.getVideoTracks()[0], null);
    ```
- |stream| indicates which stream grouping to communicate
  - If absent, a new stream is created at the receiver
  - This information is immutable; you can't change the grouping of a track (as seen by the remote side) once it has been added

Special case: onremovestream

- No longer needed after the move to tracks
- When a track is removed, it simply ends ("ended" state)
- If the track is later readded, a new track is created at the receiver
- Therefore: no **onremovetrack** event

API: The Basics

```
interface RTCRtpSender {
  readonly attribute MediaStreamTrack track;
};

interface RTCRtpReceiver {
  readonly attribute MediaStreamTrack track;
};


interface AddTrackEvent : Event {
  readonly attribute RtpReceiver receiver;
  readonly attribute MediaStreamTrack track;
  readonly attribute MediaStream stream;
};


partial interface RTCPeerConnection {
  // because a track can be part of multiple streams, the |stream| parameter
  // indicates which particular stream should be referenced in signaling
  // Fails if |track| has already been added
  RTCRtpSender addTrack(MediaStreamTrack track, optional MediaStream stream);  // replaces addStream
  void removeTrack(RTCRtpSender sender);    // replaces removeStream
  sequence<RTCRtpSender> getSenders();      // replaces getLocalStreams
  sequence<RTCRtpReceiver> getReceivers();  // replaces getRemoteStreams
  EventHandler onaddtrack;  // replaces onaddstream; event object is AddTrackEvent.
};
```

Advanced Topics

Transports

- Like RTP streams, transports are also not exposed well from PeerConnection, e.g.
  - per-transport ICE state
  - Remote DTLS certificates
- Easy to add to our object model
  - RTCRtpSender and RTCRtpReceiver add a .**transport** property

API: Transports (1.0)

```
partial interface RTCRtpSender {
  readonly attribute RTCDtlsTransport transport;
};
partial interface RTCRtpReceiver {
  readonly attribute RTCDtlsTransport transport;
};

interface RTCDtlsTransport {
 readonly attribute RTCIceConnectionState state;
 sequence<ArrayBuffer> getRemoteCertificates();
 //... more stuff later, as needed
};
```

EncodingParameters

- Now that we have RTCRtpSender, what can we do with it?
  - Read the current encoding parameters
  - Make direct changes to the track encoding
  - Some changes don't require negotiation, or none is defined:
    - e.g. changing max send bitrate
  - Changes that do require negotiation result in onnegotiationneeded:
    - e.g. pausing a MST (i.e. "hold", "a=recvonly")
  - Cannot change things that would be inconsistent with SDP
    - e.g. changing the send codec
- Any functionality that is needed must have no negotiation, or have well-defined SDP

API: EncodingParameters (1.0)

```
dictionary RTCRtpEncodingParameters {
   unsigned int       ssrc;                  // identifies the encoding; readonly
   boolean            active;                // sending or "paused/onhold"
   unsigned int       maxBitrate = null;     // maximum bits to use for this encoding
};

partial interface RTCRtpSender {
  // 1-N encodings; in the future, N can be > 1, for simulcast or layered coding
  // Each EncodingParams specifies the details of what to send (e.g. bitrate)
  sequence<RTCRtpEncodingParams> getEncodings();
  // In 1.0, only N=1 encodings are allowed. To change encodings,
  // do .get() -> change -> .set()
  void setEncodings(sequence<RTCRtpEncodingParams> encodings);
};
```