WebRTC Plans

Candidates for Implementation in 2015/2016

- Dramatically improved permissions/device selection experience (make doorhanger less fragile, options more obvious and discoverable) [Q4]
    - doorhanger don't disappear when you click elsewhere
    - full implementation of new UX
        - previews for screensharing and cameras
        - thumbnails for camera, audio indicator for mic
        - feeding up geometries so UI can highlight window you're about to share
        - switch devices in mid-call (platform support for this - target for Fx 42)
        - merging all prompts into one
    - ACTION: File a meta bug for all the biggest pain points we want to solve in Q4 (don't block on getting everything)
        - can't disappear when you click outside
        - change inputs in-call
        - ability to mute in-call
        - live detection of newly inserted devices
        - activity indicators for microphone (low hanging fruit)
- Deciding on proper measures to make screen sharing safe enough to remove from behind a whitelist, and implement them. [Q4 design, Q4/2016 implementation]
    - Firefox is on the list of things to share still (this is the big sticking issue on current design) -> surface info if a particular selection is Firefox or not ("safe" or not)
        - sharing browser is not safe, screen is not safe, tab is not safe ->leaves window and app as "safe"
- Final resolution to "IP Address Exposure" issue, based on outcome of IETF/W3C discussions (spec) [Q3/Q4]
    - Only provide relay candidates
    - is this just "switch to private browsing mode"?
    - do we need a checkbox in the prefs?
    - ACTION: File a new bug, decide what options we need to give the user, and implement (try to do all this within a month) and put this to rest
- UX for identity [Q4]
- Network/system duress detection and UI indicators [Q4 (design) & 2016]
    - figure out what we consider to be a problem
    - what do we do with the info once we decide there's a problem
        - spec work -> inform the implementation
    - ACTION: File a bug, have a discussion in the bug and on the lists, measure how easily we can know if there's a problem, design UX, and implement
- Simulcast support (spec) [Q3&Q4]
    - ImageAttr/JSEP API/etc

- Additional NAT traversal techniques (e.g., NAT-PMP, PCP) [Q4]
- “3D Camera Capture” (cf http://w3c.github.io/mediacapture-depth/) (spec, optional) [2016]
- Handling of Video Orientation RTP header (as mandated by -video- draft) (spec) [Q4]
- Telemetry dashboard for WebRTC platform (something easy to understand at a glance) [Q4&2016]
    - CPU load, broken down into categories
    - ICE success/failure, broken down into reasons and into presence/absence of TURN servers
    - A/V sync
    - Live-updated graphs (bandwidth, loss, RTT, etc)
- Stereo support for Opus (it would be nice to have some high-fi demos) [2016] (Push to see if we can get contributor help for earlier) -- **possible differentiator**
- Packaging for WebRTC standalone library, for use in other projects [2016 unless we find an major external driver/customer]
- TURN server discovery (cf draft-ietf-tram-turn-server-discovery) (spec) [2016]
- JSEP and interfaces to media work:
    - Bundle policy and multiple bundle groups (spec) [Q4]
    - MediaPipeline rework to support listen-after-offer, multiple transports, etc [Q4]
    - Multiple codecs [Q4]
    - Simultaneous support for current and pending local descriptions (eg; pending changes codecs, does an ICE restart, changes SSRCs, changes bundle, changes rtcp-mux, and so forth) (spec) [Q4]
- DataChannel updates (spec) [Q3]
- RTP Header Extensions (spec) [Q4]
    - send-time
- REMB [Q4]
- gUM constraint hookups (AEC, etc) and updating an existing capture [Q3/Q4]
    - getSupportedConstraints (spec) [Q3/Q4]
- MediaStream/PeerConnection/gUM/Stats API completion (spec)
    - MediaStream constructors/clone [Q3/Q4]
    - MediaStream add/removeTrack [Q3/Q4]
    - MediaStreamTrack attributes [Q3/Q4]
    - MediaStreamTrack capability & constraint support [Q3/Q4]
    - Policies for RTCConfiguration [Q4]
    - PeerConnection getReceivers [Q3/Q4]
    - Identity event handlers [Q4]
    - Stats [Q3/Q4]
        - Additional items to existing stats

        - Codec/MediaStream/MediaStreamTrack/Cert stat objects
        - a bunch more ICE stats
- WebRTC “Next Generation” work [2016]
    - not implementing ORTC, but we will implement whatever the working group specifies for the next gen API for WebRTC
- UDP Circuit Breakers (draft-ietf-avtcore-rtp-circuit-breakers) (spec)
    - *Note [abr]: I don’t think RMCAT is likely to be useful to implement in this timeframe.* [2016]
- Metrics via Telemetry (see https://docs.google.com/document/d/1tG1tPGX_Zcq730G-GVmI0dCPFC5rdq5PrxVCtsYTyx0/edit?usp=sharing for details)
    - Distinguish Hello calls from non-Hello calls (to help with tracking down problems and making prioritization decisions). *If we can distinguish among non-Hello services, that would be great, but there may be privacy issues with doing that.* [Q3]
    - Call setup (success rate, renegotiations) [Q3]
        - Future/needs-definition: time to accept, conversion from DataChannel or audio to a/v
    - Call type (datachannel/audio/av), duration, multistream, screensharing/capture-type
        - Future/needs-definition: end type (normal? lost far end or shutdown?), interop
    - ICE (time to connect, type used, time to end of candidates) [Q3]
    - Media (bitrate, codec, framerate, resolution, av/sync) [Q3]
        - Q4/Future: input/output delays, resolution
    - Network (loss, RTT, jitter, rexmit, recovery) [Q3]
    - MediaStreamGraph (MSG buffering level) [Q4]
    - UI (doorhanger success rates, close with answer, reopen, reject) (Florian/Firefox team) [Q4]
    - HW (cores, CPU load)[Q3]
        - Q4/future: audio mic freq, number of camera, headset
- Metrics - capturing bad in-call data [Q4]
- Canvas - pref’d on in Fx42
    - default to 30 fps
    - intermittent bugs on Win
    - crash on Mac
    - perf games with WebGL - preserve drawing buffer (synchronous callbacks)

- Automated testing improvements [Q3 & Q4]
    - terrible coverage on slow machines
- Certificate management [Q4]
    - Improves performance, removes main thread jank

Bleeding Edge Functionality that Mozilla Could Lead the Charge On

- PERC functionallity (see https://datatracker.ietf.org/doc/charter-ietf-perc/) [Q4&2016]
    - The earliest work here would be implementing draft-ietf-avtcore-srtp-ekt
- Early integration of Daala / NETVC video codec (prototyping running code would be very helpful) [2016] - *Leveraging the VP9 patch may let it be done sooner if we can find a resource*
- It could be great if we had a resource spend an iteration or two adding WebRTC identity to the FxA servers. [Q4]
    - *Note [abr]: I have some scattered thoughts on this topic that I can share with anyone who has cycles to work on it.*
- Continue pushing RTPSender/Receiver and exposing capabilities through it (spec) [Q3&Q4]

Open Issues

General things we need to take care of:

- DTMF handling (spec) [2016]
- Renaming objects/APIs to match spec (including prefix removal) (spec) [Q3/Q4]

WebAudio plans & A/V Stack redesign

Candidates for Implementation in 2015/2016

- Remaining perf issues we're behind Chrome on [Q3]
- Full-duplex Audio input/output (consolidation) [Q3&Q4/2016]
- Media constraints - enabling/disabling AEC (shared with WebRTC plans) [Q3/Q4]
- Enable support for 32kHz [Q3]
- System-level AECs [Q4/2016] - depends on full-duplex audio
- Full-system audio output as input to AEC [Q4/2016]
- Audio output device selection (cf http://w3c.github.io/mediacapture-output/) (spec) [Q4/2016]
- Multiple MSG per process (related to output selection and Deep Audio Buffering) [2016 in favor of starting Full-duplex]

- MediaStreamGraph optimization & refactoring [Q3&Q4] - may affect playback
- Audio sandbox support and optimization [Q4&2016] - affects playback
- Measuring/tracking our oranges - verify we're actually making the oranges better[Q3, Q4, 2016]
- Improved testing for audio - Verify we're getting better [Q3, Q4, 2016]
    - test bench & manual testing

Bleeding Edge Functionality that Mozilla Could Lead the Charge On

- WebAudio Workers [Q4&2016] - lots of spec work
- Deep Audio Buffering API [2016] - partly gated on multiple MSG per process

Differentiators

- PulseAudio for B2G/Android [Q4&2016] -- more expensive phones would mitigate the need for this [Maire to follow up on getting a contractor]

WebRTC spec compliance docs

Specifications Under Consideration

These are the specs that we've implicitly agreed to take on, based on the current functionality in the platform:

- http://w3c.github.io/webrtc-pc/
- http://w3c.github.io/mediacapture-main/getusermedia.html
- http://w3c.github.io/mediacapture-screen-share/
- http://www.w3.org/TR/mediacapture-fromelement/
- http://datatracker.ietf.org/doc/draft-ietf-rtcweb-jsep/?include_text=1

JSEP has a number of transitive dependencies:

- draft-ietf-mmusic-sdp-bundle-negotiation
- draft-ietf-rtcweb-alpn
- draft-ietf-rtcweb-stun-consent-freshness
- draft-ietf-rtcweb-transports
    - draft-ietf-httpbis-tunnel-protocol
- draft-schwartz-rtcweb-return

- draft-ietf-rtcweb-data-protocol
- draft-ietf-rtcweb-data-channel
- draft-ietf-mmusic-sctp-sdp
- draft-ietf-mmusic-data-channel-sdpneg
- draft-ietf-rtcweb-security-arch
- draft-ietf-rtcweb-security
- draft-ietf-mmusic-sdp-mux-attributes
- draft-ietf-mmusic-trickle-ice

WebIDL implementation key:
+ == not implemented in Firefox yet

Media Capture and Streams

MediaStream:
```
+ [ Constructor,
+ Constructor (MediaStream stream),
+ Constructor (sequence<MediaStreamTrack> tracks)]
  interface MediaStream : EventTarget {
     readonly    attribute DOMString    id;
     sequence<MediaStreamTrack> getAudioTracks ();
     sequence<MediaStreamTrack> getVideoTracks ();
     sequence<MediaStreamTrack> getTracks ();
+    MediaStreamTrack?         getTrackById (DOMString trackId);
+    void                      addTrack (MediaStreamTrack track);
+    void                      removeTrack (MediaStreamTrack track);
+    MediaStream               clone ();
+    readonly    attribute boolean      active;
+               attribute EventHandler onactive;
+               attribute EventHandler oninactive;
+               attribute EventHandler onaddtrack;
+               attribute EventHandler onremovetrack;
  };
```

MediaStreamTrack:
```
 interface MediaStreamTrack : EventTarget {
     readonly    attribute DOMString          kind;
     readonly    attribute DOMString          id;
     readonly    attribute DOMString          label;
                 attribute boolean            enabled;
```

```
+    readonly    attribute boolean              muted;
+                attribute EventHandler         onmute;
+                attribute EventHandler         onunmute;
+    readonly    attribute boolean              _readonly;
+    readonly    attribute boolean              remote;
+    readonly    attribute MediaStreamTrackState readyState;
+                attribute EventHandler         onended;
+    MediaStreamTrack       clone ();
     void                   stop ();
+    MediaTrackCapabilities getCapabilities ();
+    MediaTrackConstraints  getConstraints ();
+    MediaTrackSettings     getSettings ();
+    Promise<void> applyConstraints (MediaTrackConstraints constraints);
+               attribute EventHandler         onoverconstrained;
+    readonly   attribute boolean      isolated;
+               attribute EventHandler onisolationchange;
 };
```

MediaTrackConstraintSet:

```
 dictionary MediaTrackConstraintSet {
             ConstrainLong      width;
             ConstrainLong      height;
+            ConstrainDouble    aspectRatio;
             ConstrainDouble    frameRate;
             ConstrainDOMString facingMode;
+            ConstrainDouble    volume;
+            ConstrainLong      sampleRate;
+            ConstrainLong      sampleSize;
+            ConstrainBoolean   echoCancellation;
             ConstrainDOMString deviceId;
+            ConstrainDOMString groupId;
 };
```

MediaStreamError:

```
 interface MediaStreamError {
     readonly    attribute DOMString  name;
     readonly    attribute DOMString? message;
+    readonly    attribute DOMString? constraintName;
 };
```

MediaDevices:

```
 interface MediaDevices : EventTarget {
+                attribute EventHandler ondevicechange;
      Promise<sequence<MediaDeviceInfo>> enumerateDevices ();
+     MediaTrackSupportedConstraints getSupportedConstraints ();
      Promise<MediaStream> getUserMedia (MediaStreamConstraints constraints);
  };
```

MediaDeviceInfo:

```
 interface MediaDeviceInfo {
      readonly    attribute DOMString       deviceId;
      readonly    attribute MediaDeviceKind kind;
      readonly    attribute DOMString       label;
+     readonly    attribute DOMString       groupId;
  };
```

MediaDeviceKind:

```
 enum MediaDeviceKind {
      "audioinput",
+     "audiooutput",
      "videoinput"
  };
```

WebRTC

RTCConfiguration:

```
 dictionary RTCConfiguration {
      sequence<RTCIceServer> iceServers;
+     RTCIceTransportPolicy  iceTransportPolicy = "all";
+     RTCBundlePolicy        bundlePolicy = "balanced";
      DOMString              peerIdentity;
  };
```

RTCOfferOptions:

```
 dictionary RTCOfferOptions {
      long    offerToReceiveVideo;
      long    offerToReceiveAudio;
```

```
+    boolean voiceActivityDetection = true;
+    boolean iceRestart = false;
 };


RTCPeerConnection:
 [ Constructor (RTCConfiguration configuration)]
  interface RTCPeerConnection : EventTarget {
     Promise<RTCSessionDescription> createOffer (optional RTCOfferOptions options);
     Promise<RTCSessionDescription> createAnswer ();
     Promise<void>        setLocalDescription (RTCSessionDescription description);
     readonly    attribute RTCSessionDescription? localDescription;
     Promise<void>        setRemoteDescription (RTCSessionDescription description);
     readonly    attribute RTCSessionDescription? remoteDescription;
     readonly    attribute RTCSignalingState      signalingState;
+     void                    updateIce (RTCConfiguration configuration);
     Promise<void>               addIceCandidate (RTCIceCandidate candidate);
     readonly    attribute RTCIceGatheringState   iceGatheringState;
     readonly    attribute RTCIceConnectionState  iceConnectionState;
+     readonly    attribute boolean?              canTrickleIceCandidates;
+     RTCConfiguration            getConfiguration ();
     void                        close ();
                 attribute EventHandler          onnegotiationneeded;
                 attribute EventHandler          onicecandidate;
                 attribute EventHandler          onsignalingstatechange;
                 attribute EventHandler          oniceconnectionstatechange;
                 attribute EventHandler          onicegatheringstatechange;
     sequence<RTCRtpSender>   getSenders ();
+     sequence<RTCRtpReceiver> getReceivers ();
     RTCRtpSender        addTrack (MediaStreamTrack track, MediaStream... streams);
     void                removeTrack (RTCRtpSender sender);
-                attribute EventHandler onaddstream;
+                 attribute EventHandler ontrack;
     RTCDataChannel createDataChannel (DOMString label,
                                       optional RTCDataChannelInit dataChannelDict);
                 attribute EventHandler ondatachannel;
+     RTCDTMFSender createDTMFSender (MediaStreamTrack track);
     Promise<RTCStatsReport> getStats (MediaStreamTrack? selector);
     void setIdentityProvider (DOMString provider, optional DOMString protocol,
                               optional DOMString username);
     void getIdentityAssertion ();
     readonly    attribute RTCIdentityAssertion? peerIdentity;
+                 attribute EventHandler          onidentityresult;
```

```
+             attribute EventHandler        onpeeridentity;
+             attribute EventHandler        onidpassertionerror;
+             attribute EventHandler        onidpvalidationerror;
 };


RTCPeerState:
 enum RTCSignalingState {
     "stable",
     "have-local-offer",
     "have-remote-offer",
+     "have-local-pranswer",
+     "have-remote-pranswer",
     "closed"
 };

 enum RTCSdpType {
     "offer",
+     "pranswer",
     "answer"
 };


RTCIceConnectionState:
 enum RTCIceConnectionState {
     "new",
     "checking",
     "connected",
+     "completed",
     "failed",
+     "disconnected",
     "closed"
 };


RTCSdpError:
   interface RTCSdpError : DOMError {
+       readonly    attribute long sdpLineNumber;
   };


RTCDataChannel:
```

```
 interface RTCDataChannel : EventTarget {
     readonly    attribute DOMString          label;
     readonly    attribute boolean            ordered;
+    readonly    attribute unsigned short?    maxPacketLifeTime;
+    readonly    attribute unsigned short?    maxRetransmits;
     readonly    attribute DOMString          protocol;
+    readonly    attribute boolean            negotiated;
     readonly    attribute unsigned short     id;
     readonly    attribute RTCDataChannelState readyState;
     readonly    attribute unsigned long      bufferedAmount;
                 attribute EventHandler       onopen;
                 attribute EventHandler       onerror;
                 attribute EventHandler       onclose;
     void close ();
                 attribute EventHandler       onmessage;
                 attribute DOMString          binaryType;
     void send (DOMString data);
     void send (Blob data);
     void send (ArrayBuffer data);
     void send (ArrayBufferView data);
 };
```

WebRTC's Stats API

RTCRtpStreamStats:

```
 dictionary RTCRtpStreamStats : RTCStats {
             DOMString    ssrc;
+            DOMString    associateStatsId;
             boolean      isRemote = false;
+            DOMString    mediaTrackId;
+            DOMString    transportId;
+            DOMString    codecId;
+            unsigned long firCount;
+            unsigned long pliCount;
+            unsigned long nackCount;
+            unsigned long sliCount;
 };
```

RTCInboundStreamStats:

```
 dictionary RTCInboundRTPStreamStats : RTCRTPStreamStats {
            unsigned long      packetsReceived;
            unsigned long long bytesReceived;
            unsigned long      packetsLost;
            double             jitter;
+           double             fractionLost;
 };
```

RTCOutboundStreamStats:

```
 dictionary RTCOutboundRTPStreamStats : RTCRTPStreamStats {
            unsigned long      packetsSent;
            unsigned long long bytesSent;
+           double             targetBitrate;
            double             roundTripTime;
 };
```

RTCCodecStats:

```
+ dictionary RTCCodecStats : RTCStats {
+           unsigned long payloadType;
+           DOMString     codec;
+           unsigned long clockRate;
+           unsigned long channels;
+           DOMString     parameters;
+ };
```

RTCMediaStreamStats:

```
+ dictionary RTCMediaStreamStats : RTCStats {
+           DOMString streamIdentifier;
+           sequence  trackIds;
+ };
```

RTCMediaStreamTrackStats:

```
+ dictionary RTCMediaStreamTrackStats : RTCStats {
+           DOMString     trackIdentifier;
+           boolean       remoteSource;
+           sequence      ssrcIds;
+           unsigned long frameWidth;
+           unsigned long frameHeight;
+           double        framesPerSecond;
```

+ unsigned long framesSent;
+ unsigned long framesReceived;
+ unsigned long framesDecoded;
+ unsigned long framesDropped;
+ unsigned long framesCorrupted;
+ double audioLevel;
+ double echoReturnLoss;
+ double echoReturnLossEnhancement;
+ };

RTCPeerConnectionStats:
+ dictionary RTCPeerConnectionStats : RTCStats {
+ unsigned long dataChannelsOpened;
+ unsigned long dataChannelsClosed;
+ };

RTCDataChannelStats:
+ dictionary RTCDataChannelStats : RTCStats {
+ DOMString label;
+ DOMString protocol;
+ long datachannelid;
+ RTCDataChannelState state;
+ unsigned long messagesSent;
+ unsigned long long bytesSent;
+ unsigned long messagesReceived;
+ unsigned long long bytesReceived;
+ };

RTCTransportStats:
+ dictionary RTCTransportStats : RTCStats {
+ unsigned long long bytesSent;
+ unsigned long long bytesReceived;
+ DOMString rtcpTransportStatsId;
+ boolean activeConnection;
+ DOMString selectedCandidatePairId;
+ DOMString localCertificateId;
+ DOMString remoteCertificateId;
+ };

RTCIceCandidateAttributes:

+ dictionary RTCIceCandidateAttributes : RTCStats {
DOMString ipAddress;
long portNumber;
DOMString transport;
RTCStatsIceCandidateType candidateType;
long priority;
+ DOMString addressSourceUrl;
};

RTCIceCandidatePairStats:
dictionary RTCIceCandidatePairStats : RTCStats {
+ DOMString transportId;
DOMString localCandidateId;
DOMString remoteCandidateId;
RTCStatsIceCandidatePairState state;
unsigned long long priority;
boolean nominated;
+ boolean writable;
boolean readable;
+ unsigned long long bytesSent;
+ unsigned long long bytesReceived;
+ double roundTripTime;
+ double availableOutgoingBitrate;
+ double availableIncomingBitrate;
};

RTCCertificateStats:
+ dictionary RTCCertificateStats : RTCStats {
+ DOMString fingerprint;
+ DOMString fingerprintAlgorithm;
+ DOMString base64Certificate;
+ DOMString issuerCertificateId;
+ };

Jsep-draft and drafts it points to :

Stuff I think is a bad idea:
* (easy) support changing media type of rejected m-sections when creating a reoffer (JESP-09 sec 5.2.2 page 33)
* (easy) reoffers aren't supposed to have any codecs that weren't in the previous

remote description (this also seems like a bad requirement, 5.2.2 page 33)
* (easy) similar rules to above for RTP extensions, rtcp-fb, rtcp-mux, and rtcp-rsize
* (hard) pranswer
* (hard) DTMF

Stuff that may be an interop headache if we implement:
* (easy) UDP/TLS/RTP/SAVPF and TCP/DTLS/RTP/SAVPF
* (easy) UDP/DTLS/SCTP and TCP/DTLS/SCTP
* (medium) m-section ordering rules specified in JSEP-09 5.2.1
* (easy) c=IN IP6 ::
* (medium) latest datachannel spec
* (easy, have this behind a pref now) tmmbr

Stuff that's in the spec, but nothing will outright break if we don't do it:
* (easy) populate and handle mid in candidates (MUST level)
* (medium) ICE candidate pool (JSEP-09 3.4.4, but can't find anything about this in the webrtc spec? Is there a PR for this somewhere?)
* (medium) ICE candidate keepalive (ie; keepalives sent before ICE starts, ICE 4.1.1.4)
  - this is more likely to break if we implement ICE candidate pool
* (hard) Simultaneous support for current and pending local descriptions (eg; pending changes codecs, does an ICE restart, changes SSRCs, changes bundle, changes rtcp-mux, and so forth)
* (hard, but mostly complete) ICE TCP
* (easy?) RTX
* (?) RTCP bandwidth (I think)
* (medium) pref to turn off host or srflx candidates (SHOULD level)
  "In addition, administrators may also wish to
  control the set of ICE candidates, and so the browser SHOULD also
  allow control via local policy, with the most restrictive policy
  prevailing."
* (easy) non-bundle-only m-sections in offers need to have unique ICE credentials
* (easy) FEC (this looks optional)
* (easy) rapid RTP sync (RECOMMENDED in rtp-usage-23)
* (easy) RTP MID Header Extension (we might lose some pre-reanswer traffic when bundle renegotiation happens)
* (easy) rtcp-rsize (to what extent does webrtc.org support this?)
* (easy) maxptime
* (easy) ICE connection state should go to “completed”
* (medium) TURN binding support

Stuff that's in the spec, that might be a minor headache for webrtc services if we don't do (missing API that isn't exactly essential):
* (easy) VoiceActivityDetection

* (medium?) createOffer in states other than stable (is this question even settled? I see some language that says it is an open issue, and some language that hints it is not)
* (medium) provisional answers
* (medium, almost all work shared with the pref that controls the same thing) RTCIceTransportPolicy (MUST level)
  * Changing the policy requires support for ICE restart
* (easy) canTrickleIceCandidates
* (medium) setConfiguration
* (easy) Allow content to choose CNAME? (rtp-usage-23 11 para 6) Don't see anything about this in the w3c spec.
* (easy) rtcp-mux policy
* (hard) ICE consent freshness
* (medium) OPUS params

Stuff that could be hard to work around if we don't support:
* (medium) bundle policy
* (medium) handling of multiple bundle groups
* (easy) dummy a=rtcp attributes
* (easy) vad=off in ssrc-audio-level
* (?) Do we support RFC 6904?
* (?) CVO
* (hard) ICE restart
  (The hard part here is rollback)
* (signaling part of this isn't too hard, probably mostly code removal)
  we need to fire onaddstream as soon as we get a remote offer, instead of waiting for offer/answer to conclude

Large work items:
* MediaPipeline work. We need the following properties:
  - MediaPipelines need to be created earlier; at the very least recv pipelines when we set the local description. This is required to support the following things:
    - simultaneous support for current and pending local descriptions, since the pending descriptions are pre-answer
  - MediaPipelines need to be able to be able to listen on more than one TransportFlow (they do not need to be able to transmit on more than one)
    - simultaneous support for current and pending local descriptions when ICE restart has been offered, or when bundle renegotiation has occurred
  - MediaPipelines need to be updated when local offer is set (so we can configure new provisional transports, codecs, etc)
  - MediaPipelines need to be updated on provisional answers

* Media conduit (and webrtc.org?) work. We need the following properties:
  - We need to be able to configure media engines to accept multiple codecs and have it work properly. I seem to recall this being a problem.

* mtransport work:
  - We need to be able to have provisional components in the ICE stack
    - Needed in order to rollback an ICE restart
  - We need to be able to have components that are not attached to any stream
    - Needed for component pooling
  - ICE candidate keepalive
  - Consent freshness

* JsepSessionImpl work. We need the following things:
  - Better bundle support. Fair bit of work here between supporting multiple bundle groups, bundle policy stuff, etc.
  - Provisional answers.
  - OPUS params.
  - latest datachannel spec

Security and identity areas:

(medium) Certificate management
(easy): Mt has a patch already: Change to ECDSA certificates

Media:

DataChannels:
  65534 channels by default
  Update from upstream to support ndata
  Leverage ndata to avoid internal buffering and large xfers without chunking; start obsoleting chunking.
  Any SDP changes? SCTP/DTLS/UDP vs UDP/DTLS/SCTP etc...

Support for RTCP and header extensions as needed by the spec
  Send-time
  not spec - REMB support?

Opus options from SDP
  Backend hookups for SDP

Various GetUserMedia changes
  Backend support for changing capture options on the fly, disabling AEC in gum constraints, etc

H.264 Mode 0 fixes (STAP-A in mode 0 is wrong)