

# RtpTransceivers

at TPAC 2015

# What's been added

A 1:1 mapping between an RtpTransceiver, m-line, RtpSender, and RtpReceiver

- `PeerConnection.addTransceiver("video", {send: bool, receive: bool})`
- `PeerConnection.addTransceiver(track)`
- `PeerConnection.ontrack` has a `.transceiver`
- `PeerConnection.getTransceivers()`
- `RtpTransceiver.mid`, `.sender`, `.receiver`, `.stopped`
  - `.sender` and `.receiver` are never null/undefined, even if `{send: false}`.
- `RtpTransceiver.stop()`
  - Turns the m-line into "port 0"

# What's been removed or changed

- `offerToReceiveAudio` and `offerToReceiveVideo` are gone
  - Replaced with `PeerConnection.addTransceiver(kind, {send: false, receive: true})`
- `addTrack` may activate an `RtpSender` rather than adding one
  - And add one otherwise
- `removeTrack` deactivates an `RtpSender` rather than removing one
  - Transceiver are never removed (except perhaps because of rollback... TBD)

# Warmup Example

```
var track = null;
var sender = pc.addTransceiver("video").sender;
var reallyAnswered = false;
getUserMedia(...).then(stream => {
  track = stream.getVideoTracks()[0];
  if (reallyAnswered) {
    sender.replaceTrack(track);
  }
});
pc.createOffer().then(pc.setLocalDescription).then(signalOffer).then(answer => {
  pc.setRemoteDescription(answer);
  // At this point, ICE, DTLS, and RTP are all warming up.
});
onReallyAnswered = function() {
  reallyAnswered = true;
  if (track) {
    sender.replaceTrack(track);
  }
};
```

# Media before signalling example

```
var audio = pc.addTransceiver("audio");  
var video = pc.addTransceiver("video");  
videoTag.srcObject = new MediaStream([audio.track, video.track]);  
  
// We are ready to receive media even before an answer comes back  
pc.createOffer().then(pc.setLocalDescription);
```

# Remaining Questions to answer in the API

- When do we reuse an RtpSender?
  - Proposed rule (in the spec already): Only “reuse” an RtpSender that has never been used before.
  - More details in later slides
- Should we add an API to activate an RtpSender?
  - More details on later slides
- How do we handle rollback?
  -

# Question 1: When does addTrack reuse a sender?

This question comes up because we want the following situation to “just work”:

## Offerer:

```
pc.addTrack(...);  
pc.createOffer(...);
```

## Answerer:

```
pc.setRemoteDescription(offer);  
pc.addTrack(...); // Attach track to sender created by setRemoteDescription  
pc.createAnswer(...);
```

One sendrecv m-line. No re-offer needed.

## Option A: Reuse sender if created by setRemoteDescription

### Pros:

- Satisfies the common scenario explained on the previous slide.

### Cons:

- Calling addTrack twice would re-use the same transceiver twice, so we need to check if the track is “active” or not.



# Option B: Reuse sender if it's currently “inactive”

## Pros:

- Satisfies the common scenario

## Cons:

- If you call `addTrack`, then `removeTrack`, then `addTrack`, the remote `RtpReceiver` will be reused, which means the remote track should be also, which isn't what's intended by the sender.
- In some situations, it will cause an m-line to be reused too much, such as `pc`.  
`addTransceiver(kind, {receive: true, send: false});`  
`pc.addTrack(track);`  
(Normally you'd expect this to mean the offerer can receive 2)

# Option C: Reuse sender if it has NEVER been active

## Pros:

- Satisfies the common scenario
- Avoids pitfalls of the previous two methods.

## Cons:

- Rule is a little bit more complex.

Recommended! And already in the spec.

## Question 2: API for activating a sender?

This is relevant to the “early warmup” scenario.

### **offerer:**

```
pc.addTransceiver(kind);  
pc.createOffer();
```

### **answerer:**

```
pc.setLocalDescription(offer);  
// A separate m-line requires a re-offer!  
var sender = pc.addTransceiver(“video”);  
// Can't call pc.addTrack because we don't have a track yet.  
getUserMedia(...).then(function(stream) {  
    sender.replaceTrack(stream.getVideoTracks()[0]);  
});
```

# Option A: Add RtpSender.activate()

```
pc.ontrack = function(e) {  
  e.transceiver.sender.activate();  
  getUserMedia(...).then(function(stream) {  
    e.transceiver.sender.replaceTrack(track);  
  });  
}
```

Pros:

- Gives flexible control to the app.

Cons:

- May require renegotiation (when nothing else on RtpSender does)
- We wanted to keep SDP-isms in RtpTransceiver, not RtpSender.

## Option B: Add `RtpTransceiver.activateSender()`

```
pc.ontrack = function(e) {  
  e.transceiver.activateSender();  
  getUserMedia(...).then(function(stream) {  
    e.transceiver.sender.replaceTrack(track);  
  });  
}
```

Pros:

- Gives flexible control to the app.
- Doesn't mess up `RtpSender` with SDP-isms and renegotiation

Cons:

- Is kind of ugly and makes us want to add `activateReceiver`, `deactivateSender`, ...

# Option C: Allow `replaceTrack` to activate a sender

```
pc.ontrack = function(e) {  
  e.transceiver.sender.replaceTrack(null);  
  getUserMedia(...).then(function(stream) {  
    e.transceiver.sender.replaceTrack(track);  
  });  
}
```

## Pros:

- No new API surface
- Doesn't mess up `RtpSender` with SDP-isms and renegotiation

## Cons:

- Really kind of ugly and implicit

### Question 3: Rollback transceiver created by setRemoteDescription

```
pc.setRemoteDescription(offer); // Transceiver created  
pc.addTrack(track); // Track attached to sender  
videoTag.srcObject = new Stream([pc.getReceivers()[0].track]));
```

Now what happens to that transceiver if we do a rollback?

Is it in `pc.getTransceivers()`?

# Option A: Never remove transceivers in a rollback

## Pros:

- Simple.
- No problems with use of transceiver before rollback.

## Cons:

- This means that a rollback, even in the simplest cases, doesn't restore to a `pre-setRemoteDescription` state.
- Transceivers might pile up with lots of rollbacks.



# Option B: Always remove transceivers in a rollback

## Pros:

- Simple.
- Transceivers won't pile up.

## Cons:

- Any operations that affected the transceiver (such as `addTrack`, in the previous example) will be thrown away.

## Option C: Only remove if unmodified since setRemoteDescription

### Pros:

- Addresses both the common use cases and the corner cases.

### Cons:

- More complex.

# Option D: Don't support rollback of remote offer

Pros:

- Really simple: Avoids the problem altogether.

Cons:

- Apps can no longer rollback remote offers.

But... what's the use case of rolling back a remote offer anyway?