Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 11, 2017

J. Uberti
Google
C. Jennings
Cisco
E. Rescorla, Ed.
Mozilla

March 10, 2017

Javascript Session Establishment Protocol
draft-ietf-rtcweb-jsep-19

Abstract

This document describes the mechanisms for allowing a Javascript application to control the signaling plane of a multimedia session via the interface specified in the W3C RTCPeerConnection API, and discusses how this relates to existing signaling protocols.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document describes how the W3C WEBRTC RTCPeerConnection interface [W3C.WD-webrtc-20140617] is used to control the setup, management and teardown of a multimedia session.

1.1. General Design of JSEP

The thinking behind WebRTC call setup has been to fully specify and control the media plane, but to leave the signaling plane up to the application as much as possible. The rationale is that different applications may prefer to use different protocols, such as the existing SIP or Jingle call signaling protocols, or something custom to the particular application, perhaps for a novel use case. In this approach, the key information that needs to be exchanged is the multimedia session description, which specifies the necessary transport and media configuration information necessary to establish the media plane.

With these considerations in mind, this document describes the Javascript Session Establishment Protocol (JSEP) that allows for full control of the signaling state machine from Javascript. As described above, JSEP assumes a model in which a Javascript application executes inside a runtime containing WebRTC APIs (the "JSEP implementation"). The JSEP implementation is almost entirely divorced from the core signaling flow, which is instead handled by the Javascript making use of two interfaces: (1) passing in local and remote session descriptions and (2) interacting with the ICE state machine. The combination of the JSEP implementation and the Javascript application is referred to throughout this document as a "JSEP endpoint".

In this document, the use of JSEP is described as if it always occurs between two JSEP endpoints. Note though in many cases it will actually be between a JSEP endpoint and some kind of server, such as a gateway or MCU. This distinction is invisible to the JSEP endpoint; it just follows the instructions it is given via the API.

JSEP's handling of session descriptions is simple and straightforward. Whenever an offer/answer exchange is needed, the initiating side creates an offer by calling a createOffer() API. The application then uses that offer to set up its local config via the setLocalDescription() API. The offer is finally sent off to the remote side over its preferred signaling mechanism (e.g.,

WebSockets); upon receipt of that offer, the remote party installs it using the setRemoteDescription() API.

To complete the offer/answer exchange, the remote party uses the createAnswer() API to generate an appropriate answer, applies it using the setLocalDescription() API, and sends the answer back to the initiator over the signaling channel. When the initiator gets that answer, it installs it using the setRemoteDescription() API, and initial setup is complete. This process can be repeated for additional offer/answer exchanges.

Regarding ICE [RFC5245], JSEP decouples the ICE state machine from the overall signaling state machine, as the ICE state machine must remain in the JSEP implementation, because only the implementation has the necessary knowledge of candidates and other transport info. Performing this separation also provides additional flexibility; in protocols that decouple session descriptions from transport, such as Jingle, the session description can be sent immediately and the transport information can be sent when available. In protocols that don't, such as SIP, the information can be used in the aggregated form. Sending transport information separately can allow for faster ICE and DTLS startup, since ICE checks can start as soon as any transport information is available rather than waiting for all of it.

Through its abstraction of signaling, the JSEP approach does require the application to be aware of the signaling process. While the application does not need to understand the contents of session descriptions to set up a call, the application must call the right APIs at the right times, convert the session descriptions and ICE information into the defined messages of its chosen signaling protocol, and perform the reverse conversion on the messages it receives from the other side.

One way to mitigate this is to provide a Javascript library that hides this complexity from the developer; said library would implement a given signaling protocol along with its state machine and serialization code, presenting a higher level call-oriented interface to the application developer. For example, libraries exist to adapt the JSEP API into an API suitable for a SIP or XMPP. Thus, JSEP provides greater control for the experienced developer without forcing any additional complexity on the novice developer.

1.2. Other Approaches Considered

One approach that was considered instead of JSEP was to include a lightweight signaling protocol. Instead of providing session descriptions to the API, the API would produce and consume messages from this protocol. While providing a more high-level API, this put

more control of signaling within the JSEP implementation, forcing it to have to understand and handle concepts like signaling glare. In addition, it prevented the application from driving the state machine to a desired state, as is needed in the page reload case.

A second approach that was considered but not chosen was to decouple the management of the media control objects from session descriptions, instead offering APIs that would control each component directly. This was rejected based on a feeling that requiring exposure of this level of complexity to the application programmer would not be beneficial; it would result in an API where even a simple example would require a significant amount of code to orchestrate all the needed interactions, as well as creating a large API surface that needed to be agreed upon and documented. In addition, these API points could be called in any order, resulting in a more complex set of interactions with the media subsystem than the JSEP approach, which specifies how session descriptions are to be evaluated and applied.

One variation on JSEP that was considered was to keep the basic session description-oriented API, but to move the mechanism for generating offers and answers out of the JSEP implementation. Instead of providing createOffer/createAnswer methods within the implementation, this approach would instead expose a getCapabilities API which would provide the application with the information it needed in order to generate its own session descriptions. This increases the amount of work that the application needs to do; it needs to know how to generate session descriptions from capabilities, and especially how to generate the correct answer from an arbitrary offer and the supported capabilities. While this could certainly be addressed by using a library like the one mentioned above, it basically forces the use of said library even for a simple example. Providing createOffer/createAnswer avoids this problem.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. Semantics and Syntax

3.1. Signaling Model

JSEP does not specify a particular signaling model or state machine, other than the generic need to exchange session descriptions in the fashion described by [RFC3264](offer/answer) in order for both sides of the session to know how to conduct the session. JSEP provides

mechanisms to create offers and answers, as well as to apply them to a session. However, the JSEP implementation is totally decoupled from the actual mechanism by which these offers and answers are communicated to the remote side, including addressing, retransmission, forking, and glare handling. These issues are left entirely up to the application; the application has complete control over which offers and answers get handed to the implementation, and when.

```
    +-----------+                               +-----------+
    |  Web App  |<--- App-Specific Signaling -->|  Web App  |
    +-----------+                               +-----------+
          ^                                            ^
          |  SDP                                       |  SDP
          V                                            V
    +-----------+                                +-----------+
    |   JSEP    |<----------- Media ------------>|   JSEP    |
    |   Impl.   |                                |   Impl.   |
    +-----------+                                +-----------+
```

Figure 1: JSEP Signaling Model

3.2. Session Descriptions and State Machine

In order to establish the media plane, the user agent needs specific parameters to indicate what to transmit to the remote side, as well as how to handle the media that is received. These parameters are determined by the exchange of session descriptions in offers and answers, and there are certain details to this process that must be handled in the JSEP APIs.

Whether a session description applies to the local side or the remote side affects the meaning of that description. For example, the list of codecs sent to a remote party indicates what the local side is willing to receive, which, when intersected with the set of codecs the remote side supports, specifies what the remote side should send. However, not all parameters follow this rule; for example, the fingerprints [I-D.ietf-mmusic-4572-update] sent to a remote party are calculated based on the local certificate(s) offered; the remote party MUST either accept these parameters or reject them altogether, with no option to choose different values.

In addition, various RFCs put different conditions on the format of offers versus answers. For example, an offer may propose an arbitrary number of m= sections (i.e., media descriptions as

described in [RFC4566], Section 5.14), but an answer must contain the exact same number as the offer.

Lastly, while the exact media parameters are only known only after an offer and an answer have been exchanged, it is possible for the offerer to receive media after they have sent an offer and before they have received an answer. To properly process incoming media in this case, the offerer's media handler must be aware of the details of the offer before the answer arrives.

Therefore, in order to handle session descriptions properly, the user agent needs:

1. To know if a session description pertains to the local or remote side.

2. To know if a session description is an offer or an answer.

3. To allow the offer to be specified independently of the answer.

JSEP addresses this by adding both setLocalDescription and setRemoteDescription methods and having session description objects contain a type field indicating the type of session description being supplied. This satisfies the requirements listed above for both the offerer, who first calls setLocalDescription(sdp [offer]) and then later setRemoteDescription(sdp [answer]), as well as for the answerer, who first calls setRemoteDescription(sdp [offer]) and then later setLocalDescription(sdp [answer]).

JSEP also allows for an answer to be treated as provisional by the application. Provisional answers provide a way for an answerer to communicate initial session parameters back to the offerer, in order to allow the session to begin, while allowing a final answer to be specified later. This concept of a final answer is important to the offer/answer model; when such an answer is received, any extra resources allocated by the caller can be released, now that the exact session configuration is known. These "resources" can include things like extra ICE components, TURN candidates, or video decoders. Provisional answers, on the other hand, do no such deallocation; as a result, multiple dissimilar provisional answers, with their own codec choices, transport parameters, etc., can be received and applied during call setup. Note that the final answer itself may be different than any received provisional answers.

In [RFC3264], the constraint at the signaling level is that only one offer can be outstanding for a given session, but at the media stack level, a new offer can be generated at any point. For example, when using SIP for signaling, if one offer is sent, then cancelled using a

SIP CANCEL, another offer can be generated even though no answer was received for the first offer. To support this, the JSEP media layer can provide an offer via the createOffer() method whenever the Javascript application needs one for the signaling. The answerer can send back zero or more provisional answers, and finally end the offer-answer exchange by sending a final answer. The state machine for this is as follows:

```
                       setRemote(OFFER)               setLocal(PRANSWER)
                           /-----\                               /-----\
                           |     |                               |     |
                           v     |                               v     |
            +---------------+    |                +---------------+    |
            |               |----/                |               |----/
            |               | setLocal(PRANSWER)  |               |
            |  Remote-Offer |------------------- >| Local-Pranswer|
            |               |                     |               |
            |               |                     |               |
            +---------------+                     +---------------+
                 ^   |                                   |
                 |   | setLocal(ANSWER)                  |
   setRemote(OFFER)  |                                   |
                 |   V                  setLocal(ANSWER) |
            +---------------+                            |
            |               |                            |
            |               |<---------------------------+
            |    Stable     |
            |               |<---------------------------+
            |               |                            |
            +---------------+          setRemote(ANSWER) |
                 ^   |                                   |
                 |   | setLocal(OFFER)                   |
   setRemote(ANSWER) |                                   |
                 |   V                                   |
            +---------------+                     +---------------+
            |               |                     |               |
            |               | setRemote(PRANSWER) |               |
            |  Local-Offer  |------------------- >|Remote-Pranswer|
            |               |                     |               |
            |               |----\                |               |----\
            +---------------+    |                +---------------+    |
                           ^     |                               ^     |
                           |     |                               |     |
                           \-----/                               \-----/
                       setLocal(OFFER)               setRemote(PRANSWER)
```

Figure 2: JSEP State Machine

Aside from these state transitions there is no other difference between the handling of provisional ("pranswer") and final ("answer") answers.

3.3. Session Description Format

JSEP's session descriptions use SDP syntax for their internal representation. While this format is not optimal for manipulation from Javascript, it is widely accepted, and frequently updated with new features; any alternate encoding of session descriptions would have to keep pace with the changes to SDP, at least until the time that this new encoding eclipsed SDP in popularity.

However, to simplify Javascript processing, and provide for future flexibility, the SDP syntax is encapsulated within a SessionDescription object, which can be constructed from SDP, and be serialized out to SDP. If future specifications agree on a JSON format for session descriptions, we could easily enable this object to generate and consume that JSON.

Other methods may be added to SessionDescription in the future to simplify handling of SessionDescriptions from Javascript. In the meantime, Javascript libraries can be used to perform these manipulations.

Note that most applications should be able to treat the SessionDescriptions produced and consumed by these various API calls as opaque blobs; that is, the application will not need to read or change them.

3.4. Session Description Control

In order to give the application control over various common session parameters, JSEP provides control surfaces which tell the JSEP implementation how to generate session descriptions. This avoids the need for Javascript to modify session descriptions in most cases.

Changes to these objects result in changes to the session descriptions generated by subsequent createOffer/Answer calls.

3.4.1. RtpTransceivers

RtpTransceivers allow the application to control the RTP media associated with one m= section. Each RtpTransceiver has an RtpSender and an RtpReceiver, which an application can use to control the sending and receiving of RTP media. The application may also modify the RtpTransceiver directly, for instance, by stopping it.

RtpTransceivers generally have a 1:1 mapping with m= sections, although there may be more RtpTransceivers than m= sections when RtpTransceivers are created but not yet associated with a m= section, or if RtpTransceivers have been stopped and disassociated from m=

sections. An RtpTransceiver is said to be associated with an m= section if its mid property is non-null; otherwise it is said to be disassociated. The associated m= section is determined using a mapping between transceivers and m= section indices, formed when creating an offer or applying a remote offer. An RtpTransceiver is never associated with more than one m= section, and once a session description is applied, a m= section is always associated with exactly one RtpTransceiver.

RtpTransceivers can be created explicitly by the application or implicitly by calling setRemoteDescription with an offer that adds new m= sections.

3.4.2. RtpSenders

RtpSenders allow the application to control how RTP media is sent. An RtpSender is conceptually responsible for the outgoing RTP stream(s) described by an m= section. This includes encoding the attached MediaStreamTrack, sending RTP media packets, and generating/processing RTCP for the outgoing RTP streams(s).

3.4.3. RtpReceivers

RtpReceivers allow the application to inspect how RTP media is received. An RtpReceiver is conceptually responsible for the incoming RTP stream(s) described by an m= section. This includes processing received RTP media packets, decoding the incoming stream(s) to produce a remote MediaStreamTrack, and generating/processing RTCP for the incoming RTP stream(s).

3.5. ICE

3.5.1. ICE Gathering Overview

JSEP gathers ICE candidates as needed by the application. Collection of ICE candidates is referred to as a gathering phase, and this is triggered either by the addition of a new or recycled m= section to the local session description, or new ICE credentials in the description, indicating an ICE restart. Use of new ICE credentials can be triggered explicitly by the application, or implicitly by the JSEP implementation in response to changes in the ICE configuration.

When the ICE configuration changes in a way that requires a new gathering phase, a 'needs-ice-restart' bit is set. When this bit is set, calls to the createOffer API will generate new ICE credentials. This bit is cleared by a call to the setLocalDescription API with new ICE credentials from either an offer or an answer, i.e., from either a local- or remote-initiated ICE restart.

When a new gathering phase starts, the ICE agent will notify the application that gathering is occurring through an event. Then, when each new ICE candidate becomes available, the ICE agent will supply it to the application via an additional event; these candidates will also automatically be added to the current and/or pending local session description. Finally, when all candidates have been gathered, an event will be dispatched to signal that the gathering process is complete.

Note that gathering phases only gather the candidates needed by new/recycled/restarting m= sections; other m= sections continue to use their existing candidates. Also, when bundling is active, candidates are only gathered (and exchanged) for the m= sections referenced in BUNDLE-tags, as described in [I-D.ietf-mmusic-sdp-bundle-negotiation].

3.5.2. ICE Candidate Trickling

Candidate trickling is a technique through which a caller may incrementally provide candidates to the callee after the initial offer has been dispatched; the semantics of "Trickle ICE" are defined in [I-D.ietf-ice-trickle]. This process allows the callee to begin acting upon the call and setting up the ICE (and perhaps DTLS) connections immediately, without having to wait for the caller to gather all possible candidates. This results in faster media setup in cases where gathering is not performed prior to initiating the call.

JSEP supports optional candidate trickling by providing APIs, as described above, that provide control and feedback on the ICE candidate gathering process. Applications that support candidate trickling can send the initial offer immediately and send individual candidates when they get the notified of a new candidate; applications that do not support this feature can simply wait for the indication that gathering is complete, and then create and send their offer, with all the candidates, at this time.

Upon receipt of trickled candidates, the receiving application will supply them to its ICE agent. This triggers the ICE agent to start using the new remote candidates for connectivity checks.

3.5.2.1. ICE Candidate Format

In JSEP, ICE candidates are abstracted by an IceCandidate object, and as with session descriptions, SDP syntax is used for the internal representation.

The candidate details are specified in an IceCandidate field, using the same SDP syntax as the "candidate-attribute" field defined in [RFC5245], Section 15.1. For example:

```
candidate:1 1 UDP 1694498815 192.0.2.33 10000 typ host
```

The IceCandidate object contains a field to indicate which ICE ufrag it is associated with, as defined in [RFC5245], Section 15.4. This value is used to determine which session description (and thereby which gathering phase) this IceCandidate belongs to, which helps resolve ambiguities during ICE restarts. If this field is absent in a received IceCandidate (perhaps when communicating with a non-JSEP endpoint), the most recently received session description is assumed.

The IceCandidate object also contains fields to indicate which m= section it is associated with, which can be identified in one of two ways, either by a m= section index, or a MID. The m= section index is a zero-based index, with index N referring to the N+1th m= section in the session description referenced by this IceCandidate. The MID is a "media stream identification" value, as defined in [RFC5888], Section 4, which provides a more robust way to identify the m= section in the session description, using the MID of the associated RtpTransceiver object (which may have been locally generated by the answerer when interacting with a non-JSEP endpoint that does not support the MID attribute, as discussed in Section 5.9 below). If the MID field is present in a received IceCandidate, it MUST be used for identification; otherwise, the m= section index is used instead.

When creating an IceCandidate object, JSEP implementations MUST populate all of these fields.

3.5.3. ICE Candidate Policy

Typically, when gathering ICE candidates, the JSEP implementation will gather all possible forms of initial candidates - host, server reflexive, and relay. However, in certain cases, applications may want to have more specific control over the gathering process, due to privacy or related concerns. For example, one may want to only use relay candidates, to leak as little location information as possible (keeping in mind that this choice comes with corresponding operational costs). To accomplish this, JSEP allows the application to restrict which ICE candidates are used in a session. Note that this filtering is applied on top of any restrictions the implementation chooses to enforce regarding which IP addresses are permitted for the application, as discussed in [I-D.ietf-rtcweb-ip-handling].

There may also be cases where the application wants to change which types of candidates are used while the session is active. A prime example is where a callee may initially want to use only relay candidates, to avoid leaking location information to an arbitrary caller, but then change to use all candidates (for lower operational cost) once the user has indicated they want to take the call. For this scenario, the JSEP implementation MUST allow the candidate policy to be changed in mid-session, subject to the aforementioned interactions with local policy.

To administer the ICE candidate policy, the JSEP implementation will determine the current setting at the start of each gathering phase. Then, during the gathering phase, the implementation MUST NOT expose candidates disallowed by the current policy to the application, use them as the source of connectivity checks, or indirectly expose them via other fields, such as the raddr/rport attributes for other ICE candidates. Later, if a different policy is specified by the application, the application can apply it by kicking off a new gathering phase via an ICE restart.

3.5.4. ICE Candidate Pool

JSEP applications typically inform the JSEP implementation to begin ICE gathering via the information supplied to setLocalDescription, as this is where the app specifies the number of media streams, and thereby ICE components, for which to gather candidates. However, to accelerate cases where the application knows the number of ICE components to use ahead of time, it may ask the implementation to gather a pool of potential ICE candidates to help ensure rapid media setup.

When setLocalDescription is eventually called, and the JSEP implementation goes to gather the needed ICE candidates, it SHOULD start by checking if any candidates are available in the pool. If there are candidates in the pool, they SHOULD be handed to the application immediately via the ICE candidate event. If the pool becomes depleted, either because a larger-than-expected number of ICE components is used, or because the pool has not had enough time to gather candidates, the remaining candidates are gathered as usual. This only occurs for the first offer/answer exchange, after which the candidate pool is emptied and no longer used.

One example of where this concept is useful is an application that expects an incoming call at some point in the future, and wants to minimize the time it takes to establish connectivity, to avoid clipping of initial media. By pre-gathering candidates into the pool, it can exchange and start sending connectivity checks from these candidates almost immediately upon receipt of a call. Note

though that by holding on to these pre-gathered candidates, which will be kept alive as long as they may be needed, the application will consume resources on the STUN/TURN servers it is using.

3.6. Video Size Negotiation

Video size negotiation is the process through which a receiver can use the "a=imageattr" SDP attribute [RFC6236] to indicate what video frame sizes it is capable of receiving. A receiver may have hard limits on what its video decoder can process, or it may have some maximum set by policy.

Note that certain codecs support transmission of samples with aspect ratios other than 1.0 (i.e., non-square pixels). JSEP implementations will not transmit non-square pixels, but SHOULD receive and render such video with the correct aspect ratio. However, sample aspect ratio has no impact on the size negotiation described below; all dimensions are measured in pixels, whether square or not.

3.6.1. Creating an imageattr Attribute

The receiver will first intersect any known local limits (e.g., hardware decoder capabilities, local policy) to determine the absolute minimum and maximum sizes it can receive. If there are no known local limits, the "a=imageattr" attribute SHOULD be omitted.

Otherwise, an "a=imageattr" attribute is created with "recv" direction, and the resulting resolution space formed from the aforementioned intersection is used to specify its minimum and maximum x= and y= values. If the intersection is the null set, i.e., the degenerate case of no permitted resolutions, this MUST be represented by x=0 and y=0 values.

The rules here express a single set of preferences, and therefore, the "a=imageattr" q= value is not important. It SHOULD be set to 1.0.

The "a=imageattr" field is payload type specific. When all video codecs supported have the same capabilities, use of a single attribute, with the wildcard payload type (*), is RECOMMENDED. However, when the supported video codecs have different limitations, specific "a=imageattr" attributes MUST be inserted for each payload type.

As an example, consider a system with a multiformat video decoder, which is capable of decoding any resolution from 48x48 to 720p, In this case, the implementation would generate this attribute:

a=imageattr:* recv [x=[48:1280],y=[48:720],q=1.0]

This declaration indicates that the receiver is capable of decoding any image resolution from 48x48 up to 1280x720 pixels.

3.6.2. Interpreting an imageattr Attribute

[RFC6236] defines "a=imageattr" to be an advisory field. This means that it does not absolutely constrain the video formats that the sender can use, but gives an indication of the preferred values.

This specification prescribes more specific behavior. When a sender of a given MediaStreamTrack, which is producing video of a certain resolution, receives an "a=imageattr recv" attribute, it MUST check to see if the original resolution meets the size criteria specified in the attribute, and adapt the resolution accordingly by scaling (if appropriate). Note that when considering a MediaStreamTrack that is producing rotated video, the unrotated resolution MUST be used. This is required regardless of whether the receiver supports performing receive-side rotation (e.g., through CVO [TS26.114]), as it significantly simplifies the matching logic.

For the purposes of resolution negotiation, only size limits are considered. Any other values, e.g. picture or sample aspect ratio, MUST be ignored.

When communicating with a non-JSEP endpoint, multiple relevant "a=imageattr recv" attributes may be present in a received m= section. If this occurs, attributes other than the one with the highest "q=" value MUST be ignored. If multiple attributes have the same "q=" value, those that appear after the first such attribute in the m= section MUST be ignored.

If an "a=imageattr recv" attribute references a different video payload type than what has been selected for sending the MediaStreamTrack, it MUST be ignored.

If the original resolution matches the size limits in the attribute, the track MUST be transmitted untouched.

If the original resolution exceeds the size limits in the attribute, the sender SHOULD apply downscaling to the output of the MediaStreamTrack in order to satisfy the limits. Downscaling MUST NOT change the track aspect ratio.

If the original resolution is less than the size limits in the attribute, upscaling is needed, but this may not be appropriate in all cases. To address this concern, the application can set an

upscaling policy for each sent track. For this case, if upscaling is permitted by policy, the sender SHOULD apply upscaling in order to provide the desired resolution. Otherwise, the sender MUST NOT apply upscaling. The sender SHOULD NOT upscale in other cases, even if the policy permits it. Upscaling MUST NOT change the track aspect ratio.

If there is no appropriate and permitted scaling mechanism that allows the received size limits to be satisfied, the sender MUST NOT transmit the track.

If the attribute includes a "sar=" (sample aspect ratio) value set to something other than "1.0", indicating the receiver wants to receive non-square pixels, this cannot be satisfied and the sender MUST NOT transmit the track.

In the special case of receiving a maximum resolution of [0, 0], as described above, the sender MUST NOT transmit the track.

3.7. Simulcast

JSEP supports simulcast transmission of a MediaStreamTrack, where multiple encodings of the source media can be transmitted within the context of a single m= section. The current JSEP API is designed to allow applications to send simulcasted media but only to receive a single encoding. This allows for multi-user scenarios where each sending client sends multiple encodings to a server, which then, for each receiving client, chooses the appropriate encoding to forward.

Applications request support for simulcast by configuring multiple encodings on an RtpSender, which, upon generation of an offer or answer, are indicated in SDP markings on the corresponding m= section, as described below. Receivers that understand simulcast and are willing to receive it will also include SDP markings to indicate their support, and JSEP endpoints will use these markings to determine whether simulcast is permitted for a given RtpSender. If simulcast support is not negotiated, the RtpSender will only use the first configured encoding.

Note that the exact simulcast parameters are up to the sending application. While the aforementioned SDP markings are provided to ensure the remote side can receive and demux multiple simulcast encodings, the specific resolutions and bitrates to be used for each encoding are purely a send-side decision in JSEP.

JSEP currently does not provide a mechanism to configure receipt of simulcast. This means that if simulcast is offered by the remote endpoint, the answer generated by a JSEP endpoint will not indicate

support for receipt of simulcast, and as such the remote endpoint will only send a single encoding per m= section.

In addition, JSEP does not provide a mechanism to handle an incoming offer requesting simulcast from the JSEP endpoint. This means that established simulcast streams will continue to work through a received re-offer, but setting up initial simulcast by way of a received offer requires out-of-band signaling or SDP inspection. Future versions of this specification may add additional APIs to provide direct control.

When using JSEP to transmit multiple encodings from a RtpSender, the techniques from [I-D.ietf-mmusic-sdp-simulcast] and [I-D.ietf-mmusic-rid] are used. Specifically, when multiple encodings have been configured for a RtpSender, the m= section for the RtpSender will include an "a=simulcast" attribute, as defined in [I-D.ietf-mmusic-sdp-simulcast], Section 6.2, with a "send" simulcast stream description that lists each desired encoding, and no "recv" simulcast stream description. The m= section will also include an "a=rid" attribute for each encoding, as specified in [I-D.ietf-mmusic-rid], Section 4; the use of RID identifiers allows the individual encodings to be disambiguated even though they are all part of the same m= section.

3.8. Interactions With Forking

Some call signaling systems allow various types of forking where an SDP Offer may be provided to more than one device. For example, SIP [RFC3261] defines both a "Parallel Search" and "Sequential Search". Although these are primarily signaling level issues that are outside the scope of JSEP, they do have some impact on the configuration of the media plane that is relevant. When forking happens at the signaling layer, the Javascript application responsible for the signaling needs to make the decisions about what media should be sent or received at any point of time, as well as which remote endpoint it should communicate with; JSEP is used to make sure the media engine can make the RTP and media perform as required by the application. The basic operations that the applications can have the media engine do are:

- o Start exchanging media with a given remote peer, but keep all the resources reserved in the offer.

- o Start exchanging media with a given remote peer, and free any resources in the offer that are not being used.

3.8.1. Sequential Forking

Sequential forking involves a call being dispatched to multiple remote callees, where each callee can accept the call, but only one active session ever exists at a time; no mixing of received media is performed.

JSEP handles sequential forking well, allowing the application to easily control the policy for selecting the desired remote endpoint. When an answer arrives from one of the callees, the application can choose to apply it either as a provisional answer, leaving open the possibility of using a different answer in the future, or apply it as a final answer, ending the setup flow.

In a "first-one-wins" situation, the first answer will be applied as a final answer, and the application will reject any subsequent answers. In SIP parlance, this would be ACK + BYE.

In a "last-one-wins" situation, all answers would be applied as provisional answers, and any previous call leg will be terminated. At some point, the application will end the setup process, perhaps with a timer; at this point, the application could reapply the pending remote description as a final answer.

3.8.2. Parallel Forking

Parallel forking involves a call being dispatched to multiple remote callees, where each callee can accept the call, and multiple simultaneous active signaling sessions can be established as a result. If multiple callees send media at the same time, the possibilities for handling this are described in Section 3.1 of [RFC3960]. Most SIP devices today only support exchanging media with a single device at a time, and do not try to mix multiple early media audio sources, as that could result in a confusing situation. For example, consider having a European ringback tone mixed together with the North American ringback tone - the resulting sound would not be like either tone, and would confuse the user. If the signaling application wishes to only exchange media with one of the remote endpoints at a time, then from a media engine point of view, this is exactly like the sequential forking case.

In the parallel forking case where the Javascript application wishes to simultaneously exchange media with multiple peers, the flow is slightly more complex, but the Javascript application can follow the strategy that [RFC3960] describes using UPDATE. The UPDATE approach allows the signaling to set up a separate media flow for each peer that it wishes to exchange media with. In JSEP, this offer used in the UPDATE would be formed by simply creating a new PeerConnection

and making sure that the same local media streams have been added into this new PeerConnection. Then the new PeerConnection object would produce a SDP offer that could be used by the signaling to perform the UPDATE strategy discussed in [RFC3960].

As a result of sharing the media streams, the application will end up with N parallel PeerConnection sessions, each with a local and remote description and their own local and remote addresses. The media flow from these sessions can be managed using setDirection (see Section 4.2.3), or the application can choose to play out the media from all sessions mixed together. Of course, if the application wants to only keep a single session, it can simply terminate the sessions that it no longer needs.

4. Interface

This section details the basic operations that must be present to implement JSEP functionality. The actual API exposed in the W3C API may have somewhat different syntax, but should map easily to these concepts.

4.1. PeerConnection

4.1.1. Constructor

The PeerConnection constructor allows the application to specify global parameters for the media session, such as the STUN/TURN servers and credentials to use when gathering candidates, as well as the initial ICE candidate policy and pool size, and also the bundle policy to use.

If an ICE candidate policy is specified, it functions as described in Section 3.5.3, causing the JSEP implementation to only surface the permitted candidates (including any implementation-internal filtering) to the application, and only use those candidates for connectivity checks. The set of available policies is as follows:

all: All candidates permitted by implementation policy will be gathered and used.

relay: All candidates except relay candidates will be filtered out. This obfuscates the location information that might be ascertained by the remote peer from the received candidates. Depending on how the application deploys and chooses relay servers, this could obfuscate location to a metro or possibly even global level.

The default ICE candidate policy MUST be set to "all" as this is generally the desired policy, and also typically reduces use of application TURN server resources significantly.

If a size is specified for the ICE candidate pool, this indicates the number of ICE components to pre-gather candidates for. Because pre-gathering results in utilizing STUN/TURN server resources for potentially long periods of time, this must only occur upon application request, and therefore the default candidate pool size MUST be zero.

The application can specify its preferred policy regarding use of bundle, the multiplexing mechanism defined in [I-D.ietf-mmusic-sdp-bundle-negotiation]. Regardless of policy, the application will always try to negotiate bundle onto a single transport, and will offer a single bundle group across all m= sections; use of this single transport is contingent upon the answerer accepting bundle. However, by specifying a policy from the list below, the application can control exactly how aggressively it will try to bundle media streams together, which affects how it will interoperate with a non-bundle-aware endpoint. When negotiating with a non-bundle-aware endpoint, only the streams not marked as bundle-only streams will be established.

The set of available policies is as follows:

balanced: The first m= section of each type (audio, video, or application) will contain transport parameters, which will allow an answerer to unbundle that section. The second and any subsequent m= section of each type will be marked bundle-only. The result is that if there are N distinct media types, then candidates will be gathered for for N media streams. This policy balances desire to multiplex with the need to ensure basic audio and video can still be negotiated in legacy cases. When acting as answerer, if there is no bundle group in the offer, the implementation will reject all but the first m= section of each type.

max-compat: All m= sections will contain transport parameters; none will be marked as bundle-only. This policy will allow all streams to be received by non-bundle-aware endpoints, but require separate candidates to be gathered for each media stream.

max-bundle: Only the first m= section will contain transport parameters; all streams other than the first will be marked as bundle-only. This policy aims to minimize candidate gathering and maximize multiplexing, at the cost of less compatibility with legacy endpoints. When acting as answerer, the implementation will reject any m= sections other than the first m= section, unless they are in the same bundle group as that m= section.

As it provides the best tradeoff between performance and compatibility with legacy endpoints, the default bundle policy MUST be set to "balanced".

The application can specify its preferred policy regarding use of RTP/RTCP multiplexing [RFC5761] using one of the following policies:

negotiate: The JSEP implementation will gather both RTP and RTCP candidates but also will offer "a=rtcp-mux", thus allowing for compatibility with either multiplexing or non-multiplexing endpoints.

require: The JSEP implementation will only gather RTP candidates and will insert an "a=rtcp-mux-only" indication into any new m= sections in offers it generates. This halves the number of candidates that the offerer needs to gather. Applying a description with an m= section that does not contain an "a=rtcp-mux" attribute will cause an error to be returned.

The default multiplexing policy MUST be set to "require". Implementations MAY choose to reject attempts by the application to set the multiplexing policy to "negotiate".

4.1.2. addTrack

The addTrack method adds a MediaStreamTrack to the PeerConnection, using the MediaStream argument to associate the track with other tracks in the same MediaStream, so that they can be added to the same "LS" group when creating an offer or answer. addTrack attempts to minimize the number of transceivers as follows: If the PeerConnection is in the "have-remote-offer" state, the track will be attached to the first compatible transceiver that was created by the most recent call to setRemoteDescription() and does not have a local track. Otherwise, a new transceiver will be created, as described in Section 4.1.4.

4.1.3. removeTrack

The removeTrack method removes a MediaStreamTrack from the PeerConnection, using the RtpSender argument to indicate which sender should have its track removed. The sender's track is cleared, and the sender stops sending. Future calls to createOffer will mark the m= section associated with the sender as recvonly (if transceiver.currentDirection is sendrecv) or as inactive (if transceiver.currentDirection is sendonly).

4.1.4. addTransceiver

The addTransceiver method adds a new RtpTransceiver to the PeerConnection. If a MediaStreamTrack argument is provided, then the transceiver will be configured with that media type and the track will be attached to the transceiver. Otherwise, the application MUST explicitly specify the type; this mode is useful for creating recvonly transceivers as well as for creating transceivers to which a track can be attached at some later point.

At the time of creation, the application can also specify a transceiver direction attribute, a set of MediaStreams which the transceiver is associated with (allowing LS group assignments), and a set of encodings for the media (used for simulcast as described in Section 3.7).

4.1.5. createDataChannel

The createDataChannel method creates a new data channel and attaches it to the PeerConnection. If no data channel currently exists for this PeerConnection, then a new offer/answer exchange is required. All data channels on a given PeerConnection share the same SCTP/DTLS association and therefore the same m= section, so subsequent creation of data channels does not have any impact on the JSEP state.

The createDataChannel method also includes a number of arguments which are used by the PeerConnection (e.g., maxPacketLifetime) but are not reflected in the SDP and do not affect the JSEP state.

4.1.6. createOffer

The createOffer method generates a blob of SDP that contains a [RFC3264] offer with the supported configurations for the session, including descriptions of the media added to this PeerConnection, the codec/RTP/RTCP options supported by this implementation, and any candidates that have been gathered by the ICE agent. An options parameter may be supplied to provide additional control over the generated offer. This options parameter allows an application to

trigger an ICE restart, for the purpose of reestablishing connectivity.

In the initial offer, the generated SDP will contain all desired functionality for the session (functionality that is supported but not desired by default may be omitted); for each SDP line, the generation of the SDP will follow the process defined for generating an initial offer from the document that specifies the given SDP line. The exact handling of initial offer generation is detailed in Section 5.2.1 below.

In the event createOffer is called after the session is established, createOffer will generate an offer to modify the current session based on any changes that have been made to the session, e.g., adding or stopping RtpTransceivers, or requesting an ICE restart. For each existing stream, the generation of each SDP line must follow the process defined for generating an updated offer from the RFC that specifies the given SDP line. For each new stream, the generation of the SDP must follow the process of generating an initial offer, as mentioned above. If no changes have been made, or for SDP lines that are unaffected by the requested changes, the offer will only contain the parameters negotiated by the last offer-answer exchange. The exact handling of subsequent offer generation is detailed in Section 5.2.2. below.

Session descriptions generated by createOffer must be immediately usable by setLocalDescription; if a system has limited resources (e.g. a finite number of decoders), createOffer should return an offer that reflects the current state of the system, so that setLocalDescription will succeed when it attempts to acquire those resources.

Calling this method may do things such as generating new ICE credentials, but does not result in candidate gathering, or cause media to start or stop flowing. Specifically, the offer is not applied, and does not become the pending local description, until setLocalDescription is called.

4.1.7. createAnswer

The createAnswer method generates a blob of SDP that contains a [RFC3264] SDP answer with the supported configuration for the session that is compatible with the parameters supplied in the most recent call to setRemoteDescription, which MUST have been called prior to calling createAnswer. Like createOffer, the returned blob contains descriptions of the media added to this PeerConnection, the codec/RTP/RTCP options negotiated for this session, and any candidates that have been gathered by the ICE agent. An options

parameter may be supplied to provide additional control over the generated answer.

As an answer, the generated SDP will contain a specific configuration that specifies how the media plane should be established; for each SDP line, the generation of the SDP must follow the process defined for generating an answer from the document that specifies the given SDP line. The exact handling of answer generation is detailed in Section 5.3. below.

Session descriptions generated by createAnswer must be immediately usable by setLocalDescription; like createOffer, the returned description should reflect the current state of the system.

Calling this method may do things such as generating new ICE credentials, but does not trigger candidate gathering or cause a media state change. Specifically, the answer is not applied, and does not become the pending local description, until setLocalDescription is called.

4.1.8. SessionDescriptionType

Session description objects (RTCSessionDescription) may be of type "offer", "pranswer", "answer" or "rollback". These types provide information as to how the description parameter should be parsed, and how the media state should be changed.

"offer" indicates that a description should be parsed as an offer; said description may include many possible media configurations. A description used as an "offer" may be applied anytime the PeerConnection is in a stable state, or as an update to a previously supplied but unanswered "offer".

"pranswer" indicates that a description should be parsed as an answer, but not a final answer, and so should not result in the freeing of allocated resources. It may result in the start of media transmission, if the answer does not specify an inactive media direction. A description used as a "pranswer" may be applied as a response to an "offer", or an update to a previously sent "pranswer".

"answer" indicates that a description should be parsed as an answer, the offer-answer exchange should be considered complete, and any resources (decoders, candidates) that are no longer needed can be released. A description used as an "answer" may be applied as a response to an "offer", or an update to a previously sent "pranswer".

The only difference between a provisional and final answer is that the final answer results in the freeing of any unused resources that

were allocated as a result of the offer. As such, the application can use some discretion on whether an answer should be applied as provisional or final, and can change the type of the session description as needed. For example, in a serial forking scenario, an application may receive multiple "final" answers, one from each remote endpoint. The application could choose to accept the initial answers as provisional answers, and only apply an answer as final when it receives one that meets its criteria (e.g. a live user instead of voicemail).

"rollback" is a special session description type implying that the state machine should be rolled back to the previous stable state, as described in Section 4.1.8.2. The contents MUST be empty.

4.1.8.1. Use of Provisional Answers

Most applications will not need to create answers using the "pranswer" type. While it is good practice to send an immediate response to an offer, in order to warm up the session transport and prevent media clipping, the preferred handling for a JSEP application is to create and send a "sendonly" final answer with a null MediaStreamTrack immediately after receiving the offer, which will prevent media from being sent by the caller, and allow media to be sent immediately upon answer by the callee. Later, when the callee actually accepts the call, the application can plug in the real MediaStreamTrack and create a new "sendrecv" offer to update the previous offer/answer pair and start bidirectional media flow. While this could also be done with a "sendonly" pranswer, followed by a "sendrecv" answer, the initial pranswer leaves the offer-answer exchange open, which means that the caller cannot send an updated offer during this time.

As an example, consider a typical JSEP application that wants to set up audio and video as quickly as possible. When the callee receives an offer with audio and video MediaStreamTracks, it will send an immediate answer accepting these tracks as sendonly (meaning that the caller will not send the callee any media yet, and because the callee has not yet added its own MediaStreamTracks, the callee will not send any media either). It will then ask the user to accept the call and acquire the needed local tracks. Upon acceptance by the user, the application will plug in the tracks it has acquired, which, because ICE and DTLS handshaking have likely completed by this point, can start transmitting immediately. The application will also send a new offer to the remote side indicating call acceptance and moving the audio and video to be two-way media. A detailed example flow along these lines is shown in Section 7.3.

Of course, some applications may not be able to perform this double offer-answer exchange, particularly ones that are attempting to gateway to legacy signaling protocols. In these cases, pranswer can still provide the application with a mechanism to warm up the transport.

4.1.8.2. Rollback

In certain situations it may be desirable to "undo" a change made to setLocalDescription or setRemoteDescription. Consider a case where a call is ongoing, and one side wants to change some of the session parameters; that side generates an updated offer and then calls setLocalDescription. However, the remote side, either before or after setRemoteDescription, decides it does not want to accept the new parameters, and sends a reject message back to the offerer. Now, the offerer, and possibly the answerer as well, need to return to a stable state and the previous local/remote description. To support this, we introduce the concept of "rollback".

A rollback discards any proposed changes to the session, returning the state machine to the stable state, and setting the pending local and/or remote description (see Section 4.1.12 and Section 4.1.14) to null. Any resources or candidates that were allocated by the abandoned local description are discarded; any media that is received will be processed according to the previous local and remote descriptions. Rollback can only be used to cancel proposed changes; there is no support for rolling back from a stable state to a previous stable state. Note that this implies that once the answerer has performed setLocalDescription with his answer, this cannot be rolled back.

A rollback will disassociate any RtpTransceivers that were associated with m= sections by the application of the rolled-back session description (see Section 5.9 and Section 5.8). This means that some RtpTransceivers that were previously associated will no longer be associated with any m= section; in such cases, the value of the RtpTransceiver's mid property MUST be set to null, and the mapping between the transceiver and its m= section index MUST be discarded. RtpTransceivers that were created by applying a remote offer that was subsequently rolled back MUST be stopped and removed from the PeerConnection. However, a RtpTransceiver MUST NOT be removed if a track was attached to the RtpTransceiver via the addTrack method. This is so that an application may call addTrack, then call setRemoteDescription with an offer, then roll back that offer, then call createOffer and have a m= section for the added track appear in the generated offer.

A rollback is performed by supplying a session description of type "rollback" with empty contents to either setLocalDescription or setRemoteDescription, depending on which was most recently used (i.e. if the new offer was supplied to setLocalDescription, the rollback should be done using setLocalDescription as well).

4.1.9. setLocalDescription

The setLocalDescription method instructs the PeerConnection to apply the supplied session description as its local configuration. The type field indicates whether the description should be processed as an offer, provisional answer, or final answer; offers and answers are checked differently, using the various rules that exist for each SDP line.

This API changes the local media state; among other things, it sets up local resources for receiving and decoding media. In order to successfully handle scenarios where the application wants to offer to change from one media format to a different, incompatible format, the PeerConnection must be able to simultaneously support use of both the current and pending local descriptions (e.g., support the codecs that exist in either description). This dual processing begins when the PeerConnection enters the have-local-offer state, and continues until setRemoteDescription is called with either a final answer, at which point the PeerConnection can fully adopt the pending local description, or a rollback, which results in a revert to the current local description.

This API indirectly controls the candidate gathering process. When a local description is supplied, and the number of transports currently in use does not match the number of transports needed by the local description, the PeerConnection will create transports as needed and begin gathering candidates for each transport, using ones from the candidate pool if available.

If setRemoteDescription was previously called with an offer, and setLocalDescription is called with an answer (provisional or final), and the media directions are compatible, and media is available to send, this will result in the starting of media transmission.

4.1.10. setRemoteDescription

The setRemoteDescription method instructs the PeerConnection to apply the supplied session description as the desired remote configuration. As in setLocalDescription, the type field of the description indicates how it should be processed.

This API changes the local media state; among other things, it sets up local resources for sending and encoding media.

If setLocalDescription was previously called with an offer, and setRemoteDescription is called with an answer (provisional or final), and the media directions are compatible, and media is available to send, this will result in the starting of media transmission.

4.1.11. currentLocalDescription

The currentLocalDescription method returns the current negotiated local description - i.e., the local description from the last successful offer/answer exchange - in addition to any local candidates that have been generated by the ICE agent since the local description was set.

A null object will be returned if an offer/answer exchange has not yet been completed.

4.1.12. pendingLocalDescription

The pendingLocalDescription method returns a copy of the local description currently in negotiation - i.e., a local offer set without any corresponding remote answer - in addition to any local candidates that have been generated by the ICE agent since the local description was set.

A null object will be returned if the state of the PeerConnection is "stable" or "have-remote-offer".

4.1.13. currentRemoteDescription

The currentRemoteDescription method returns a copy of the current negotiated remote description - i.e., the remote description from the last successful offer/answer exchange - in addition to any remote candidates that have been supplied via processIceMessage since the remote description was set.

A null object will be returned if an offer/answer exchange has not yet been completed.

4.1.14. pendingRemoteDescription

The pendingRemoteDescription method returns a copy of the remote description currently in negotiation - i.e., a remote offer set without any corresponding local answer - in addition to any remote candidates that have been supplied via processIceMessage since the remote description was set.

A null object will be returned if the state of the PeerConnection is "stable" or "have-local-offer".

4.1.15. canTrickleIceCandidates

The canTrickleIceCandidates property indicates whether the remote side supports receiving trickled candidates. There are three potential values:

null: No SDP has been received from the other side, so it is not known if it can handle trickle. This is the initial value before setRemoteDescription() is called.

true: SDP has been received from the other side indicating that it can support trickle.

false: SDP has been received from the other side indicating that it cannot support trickle.

As described in Section 3.5.2, JSEP implementations always provide candidates to the application individually, consistent with what is needed for Trickle ICE. However, applications can use the canTrickleIceCandidates property to determine whether their peer can actually do Trickle ICE, i.e., whether it is safe to send an initial offer or answer followed later by candidates as they are gathered. As "true" is the only value that definitively indicates remote Trickle ICE support, an application which compares canTrickleIceCandidates against "true" will by default attempt Half Trickle on initial offers and Full Trickle on subsequent interactions with a Trickle ICE-compatible agent.

4.1.16. setConfiguration

The setConfiguration method allows the global configuration of the PeerConnection, which was initially set by constructor parameters, to be changed during the session. The effects of this method call depend on when it is invoked, and differ depending on which specific parameters are changed:

o Any changes to the STUN/TURN servers to use affect the next gathering phase. If an ICE gathering phase has already started or completed, the 'needs-ice-restart' bit mentioned in Section 3.5.1 will be set. This will cause the next call to createOffer to generate new ICE credentials, for the purpose of forcing an ICE restart and kicking off a new gathering phase, in which the new servers will be used. If the ICE candidate pool has a nonzero size, and a local description has not yet been applied, any

existing candidates will be discarded, and new candidates will be gathered from the new servers.

- o Any change to the ICE candidate policy affects the next gathering phase. If an ICE gathering phase has already started or completed, the 'needs-ice-restart' bit will be set. Either way, changes to the policy have no effect on the candidate pool, because pooled candidates are not surfaced to the application until a gathering phase occurs, and so any necessary filtering can still be done on any pooled candidates.

- o The ICE candidate pool size MUST NOT be changed after applying a local description. If a local description has not yet been applied, any changes to the ICE candidate pool size take effect immediately; if increased, additional candidates are pre-gathered; if decreased, the now-superfluous candidates are discarded.

- o The bundle and RTCP-multiplexing policies MUST NOT be changed after the construction of the PeerConnection.

This call may result in a change to the state of the ICE Agent.

4.1.17. addIceCandidate

The addIceCandidate method provides a remote candidate to the ICE agent, which, if parsed successfully, will be added to the current and/or pending remote description according to the rules defined for Trickle ICE. The pair of MID and ufrag is used to determine the m= section and ICE candidate generation to which the candidate belongs. If the MID is not present, the m= section index is used to look up the locally generated MID (see Section 5.9), which is used in place of a supplied MID. If these values or the candidate string are invalid, an error is generated.

The purpose of the ufrag is to resolve ambiguities when trickle ICE is in progress during an ICE restart. If the ufrag is absent, the candidate MUST be assumed to belong to the most recently applied remote description. Connectivity checks will be sent to the new candidate.

This method can also be used to provide an end-of-candidates indication to the ICE agent, as defined in [I-D.ietf-ice-trickle]). The MID and ufrag are used as described above to determine the m= section and ICE generation for which candidate gathering is complete. If the ufrag is not present, then the end-of-candidates indication MUST be assumed to apply to the relevant m= section in the most recently applied remote description. If neither the MID nor the m=

index is present, then the indication MUST be assumed to apply to all m= sections in the most recently applied remote description.

This call will result in a change to the state of the ICE Agent, and may result in a change to media state if it results in connectivity being established.

4.2. RtpTransceiver

4.2.1. stop

The stop method stops an RtpTransceiver. This will cause future calls to createOffer to generate a zero port for the associated m= section. See below for more details.

4.2.2. stopped

The stopped property indicates whether the transceiver has been stopped, either by a call to stopTransceiver or by applying an answer that rejects the associated m= section. In either of these cases, it is set to "true", and otherwise will be set to "false".

A stopped RtpTransceiver does not send any outgoing RTP or RTCP or process any incoming RTP or RTCP. It cannot be restarted.

4.2.3. setDirection

The setDirection method sets the direction of a transceiver, which affects the direction property of the associated m= section on future calls to createOffer and createAnswer.

When creating offers, the transceiver direction is directly reflected in the output, even for reoffers. When creating answers, the transceiver direction is intersected with the offered direction, as explained in the Section 5.3 section below.

Note that while setDirection sets the direction property of the transceiver immediately (Section 4.2.4), this property does not immediately affect whether the transceiver's RtpSender will send or its RtpReceiver will receive. The direction in effect is represented by the currentDirection property, which is only updated when an answer is applied.

4.2.4. direction

The direction property indicates the last value passed into setDirection. If setDirection has never been called, it is set to the direction the transceiver was initialized with.

4.2.5. currentDirection

The currentDirection property indicates the last negotiated direction for the transceiver's associated m= section. More specifically, it indicates the [RFC3264] directional attribute of the associated m= section in the last applied answer, with "send" and "recv" directions reversed if it was a remote answer. For example, if the directional attribute for the associated m= section in a remote answer is "recvonly", currentDirection is set to "sendonly".

If an answer that references this transceiver has not yet been applied, or if the transceiver is stopped, currentDirection is set to null.

4.2.6. setCodecPreferences

The setCodecPreferences method sets the codec preferences of a transceiver, which in turn affect the presence and order of codecs of the associated m= section on future calls to createOffer and createAnswer. Note that setCodecPreferences does not directly affect which codec the implementation decides to send. It only affects which codecs the implementation indicates that it prefers to receive, via the offer or answer. Even when a codec is excluded by setCodecPreferences, it still may be used to send until the next offer/answer exchange discards it.

The codec preferences of an RtpTransceiver can cause codecs to be excluded by subsequent calls to createOffer and createAnswer, in which case the corresponding media formats in the associated m= section will be excluded. The codec preferences cannot add media formats that would otherwise not be present. This includes codecs that were not negotiated in a previous offer/answer exchange that included the transceiver.

The codec preferences of an RtpTransceiver can also determine the order of codecs in subsequent calls to createOffer and createAnswer, in which case the order of the media formats in the associated m= section will follow the specified preferences.

5. SDP Interaction Procedures

This section describes the specific procedures to be followed when creating and parsing SDP objects.

5.1. Requirements Overview

JSEP implementations must comply with the specifications listed below that govern the creation and processing of offers and answers.

5.1.1. Usage Requirements

All session descriptions handled by JSEP implementations, both local and remote, MUST indicate support for the following specifications. If any of these are absent, this omission MUST be treated as an error.

- ICE, as specified in [RFC5245], MUST be used. Note that the remote endpoint may use a Lite implementation; implementations MUST properly handle remote endpoints which do ICE-Lite.

- DTLS [RFC6347] or DTLS-SRTP [RFC5763], MUST be used, as appropriate for the media type, as specified in [I-D.ietf-rtcweb-security-arch]

The SDES SRTP keying mechanism from [RFC4568] MUST NOT be used, as discussed in [I-D.ietf-rtcweb-security-arch].

5.1.2. Profile Names and Interoperability

For media m= sections, JSEP implementations MUST support the "UDP/TLS/RTP/SAVPF" profile specified in [RFC7850], and MUST indicate this profile for each media m= line they produce in an offer. For data m= sections, implementations MUST support the "UDP/DTLS/SCTP" profile and MUST indicate this profile for each data m= line they produce in an offer. Because ICE can select either UDP [RFC5245] or TCP [RFC6544] transport depending on network conditions, this advertisement is consistent with ICE eventually selecting either either UDP or TCP.

Unfortunately, in an attempt at compatibility, some endpoints generate other profile strings even when they mean to support one of these profiles. For instance, an endpoint might generate "RTP/AVP" but supply "a=fingerprint" and "a=rtcp-fb" attributes, indicating its willingness to support "(UDP,TCP)/TLS/RTP/SAVPF". In order to simplify compatibility with such endpoints, JSEP implementations MUST follow the following rules when processing the media m= sections in an offer:

- The profile in any "m=" line in any answer MUST exactly match the profile provided in the offer.

- Any profile matching the following patterns MUST be accepted: "RTP/[S]AVP[F]" and "(UDP/TCP)/TLS/RTP/SAVP[F]"

- Because DTLS-SRTP is REQUIRED, the choice of SAVP or AVP has no effect; support for DTLS-SRTP is determined by the presence of one or more "a=fingerprint" attribute. Note that lack of an "a=fingerprint" attribute will lead to negotiation failure.

- The use of AVPF or AVP simply controls the timing rules used for RTCP feedback. If AVPF is provided, or an "a=rtcp-fb" attribute is present, assume AVPF timing, i.e., a default value of "trr-int=0". Otherwise, assume that AVPF is being used in an AVP compatible mode and use a value of "trr-int=4000".

- For data m= sections, implementations MUST support receiving the "UDP/DTLS/SCTP", "TCP/DTLS/SCTP", or "DTLS/SCTP" (for backwards compatibility) profiles.

Note that re-offers by JSEP implementations MUST use the correct profile strings even if the initial offer/answer exchange used an (incorrect) older profile string.

5.2. Constructing an Offer

When createOffer is called, a new SDP description must be created that includes the functionality specified in [I-D.ietf-rtcweb-rtp-usage]. The exact details of this process are explained below.

5.2.1. Initial Offers

When createOffer is called for the first time, the result is known as the initial offer.

The first step in generating an initial offer is to generate session-level attributes, as specified in [RFC4566], Section 5. Specifically:

- The first SDP line MUST be "v=0", as specified in [RFC4566], Section 5.1

- The second SDP line MUST be an "o=" line, as specified in [RFC4566], Section 5.2. The value of the <username> field SHOULD be "-". [RFC3264] requires that the <sess-id> be representable as a 64-bit signed integer. It is RECOMMENDED that the <sess-id> be generated as a 64-bit quantity with the high bit being sent to zero and the remaining 63 bits being cryptographically random. The value of the <nettype> <addrtype> <unicast-address> tuple

SHOULD be set to a non-meaningful address, such as IN IP4 0.0.0.0, to prevent leaking the local address in this field. As mentioned in [RFC4566], the entire o= line needs to be unique, but selecting a random number for <sess-id> is sufficient to accomplish this.

- The third SDP line MUST be a "s=" line, as specified in [RFC4566], Section 5.3; to match the "o=" line, a single dash SHOULD be used as the session name, e.g. "s=-". Note that this differs from the advice in [RFC4566] which proposes a single space, but as both "o=" and "s=" are meaningless, having the same meaningless value seems clearer.

- Session Information ("i="), URI ("u="), Email Address ("e="), Phone Number ("p="), Repeat Times ("r="), and Time Zones ("z=") lines are not useful in this context and SHOULD NOT be included.

- Encryption Keys ("k=") lines do not provide sufficient security and MUST NOT be included.

- A "t=" line MUST be added, as specified in [RFC4566], Section 5.9; both <start-time> and <stop-time> SHOULD be set to zero, e.g. "t=0 0".

- An "a=ice-options" line with the "trickle" option MUST be added, as specified in [I-D.ietf-ice-trickle], Section 4.

- If WebRTC identity is being used, an "a=identity" line as described in [I-D.ietf-rtcweb-security-arch], Section 5.

The next step is to generate m= sections, as specified in [RFC4566] Section 5.14. An m= section is generated for each RtpTransceiver that has been added to the PeerConnection, excluding any stopped RtpTransceivers. This is done in the order the RtpTransceivers were added to the PeerConnection.

For each m= section generated for an RtpTransceiver, establish a mapping between the transceiver and the index of the generated m= section.

Each m= section, provided it is not marked as bundle-only, MUST generate a unique set of ICE credentials and gather its own unique set of ICE candidates. Bundle-only m= sections MUST NOT contain any ICE credentials and MUST NOT gather any candidates.

For DTLS, all m= sections MUST use all the certificate(s) that have been specified for the PeerConnection; as a result, they MUST all have the same [I-D.ietf-mmusic-4572-update] fingerprint value(s), or these value(s) MUST be session-level attributes.

Each m= section should be generated as specified in [RFC4566], Section 5.14. For the m= line itself, the following rules MUST be followed:

o The port value is set to the port of the default ICE candidate for this m= section, but given that no candidates are available yet, the "dummy" port value of 9 (Discard) MUST be used, as indicated in [I-D.ietf-ice-trickle], Section 5.1.

o To properly indicate use of DTLS, the <proto> field MUST be set to "UDP/TLS/RTP/SAVPF", as specified in [RFC5764], Section 8.

o If codec preferences have been set for the associated transceiver, media formats MUST be generated in the corresponding order, and MUST exclude any codecs not present in the codec preferences.

o The media formats in the answer MAY include codecs present in the offer that were discarded in a previous offer/answer exchange. This is necessary for compatibility with third- party call control and SIP use cases.

o Unless excluded by the above restrictions, the media formats MUST include the mandatory audio/video codecs as specified in [I-D.ietf-rtcweb-audio](see Section 3) and [I-D.ietf-rtcweb-video](see Section 5).

The m= line MUST be followed immediately by a "c=" line, as specified in [RFC4566], Section 5.7. Again, as no candidates are available yet, the "c=" line must contain the "dummy" value "IN IP4 0.0.0.0", as defined in [I-D.ietf-ice-trickle], Section 5.1.

[I-D.ietf-mmusic-sdp-mux-attributes] groups SDP attributes into different categories. To avoid unnecessary duplication when bundling, Section 8.1 of [I-D.ietf-mmusic-sdp-bundle-negotiation] specifies that attributes of category IDENTICAL or TRANSPORT should not be repeated in bundled m= sections.

The following attributes, which are of a category other than IDENTICAL or TRANSPORT, MUST be included in each m= section:

o An "a=mid" line, as specified in [RFC5888], Section 4. All MID values MUST be generated in a fashion that does not leak user information, e.g., randomly or using a per-PeerConnection counter, and SHOULD be 3 bytes or less, to allow them to efficiently fit into the RTP header extension defined in [I-D.ietf-mmusic-sdp-bundle-negotiation], Section 14. Note that this does not set the RtpTransceiver mid property, as that only

occurs when the description is applied. The generated MID value can be considered a "proposed" MID at this point.

- A direction attribute which is the same as that of the associated transceiver.

- For each media format on the m= line, "a=rtpmap" and "a=fmtp" lines, as specified in [RFC4566], Section 6, and [RFC3264], Section 5.1.

- For each primary codec where RTP retransmission should be used, a corresponding "a=rtpmap" line indicating "rtx" with the clock rate of the primary codec and an "a=fmtp" line that references the payload type of the primary codec, as specified in [RFC4588], Section 8.1.

- For each supported FEC mechanism, "a=rtpmap" and "a=fmtp" lines, as specified in [RFC4566], Section 6. The FEC mechanisms that MUST be supported are specified in [I-D.ietf-rtcweb-fec], Section 6, and specific usage for each media type is outlined in Sections 4 and 5.

- If this m= section is for media with configurable durations of media per packet, e.g., audio, an "a=maxptime" line, indicating the maximum amount of media, specified in milliseconds, that can be encapsulated in each packet, as specified in [RFC4566], Section 6. This value is set to the smallest of the maximum duration values across all the codecs included in the m= section.

- If this m= section is for video media, and there are known limitations on the size of images which can be decoded, an "a=imageattr" line, as specified in Section 3.6.

- For each supported RTP header extension, an "a=extmap" line, as specified in [RFC5285], Section 5. The list of header extensions that SHOULD/MUST be supported is specified in [I-D.ietf-rtcweb-rtp-usage], Section 5.2. Any header extensions that require encryption MUST be specified as indicated in [RFC6904], Section 4.

- For each supported RTCP feedback mechanism, an "a=rtcp-fb" mechanism, as specified in [RFC4585], Section 4.2. The list of RTCP feedback mechanisms that SHOULD/MUST be supported is specified in [I-D.ietf-rtcweb-rtp-usage], Section 5.1.

- If the RtpTransceiver has a sendrecv or sendonly direction:

* For each MediaStream that was associated with the transceiver when it was created via addTrack or addTransceiver, an "a=msid" line, as specified in [I-D.ietf-mmusic-msid], Section 2. If a MediaStreamTrack is attached to the transceiver's RtpSender, the "a=msid" lines MUST use that track's ID. If no MediaStreamTrack is attached, a valid ID MUST be generated, in the same way that the implementation generates IDs for local tracks.

* If no MediaStream is associated with the transceiver, a single "a=msid" line with the special value "-" in place of the MediaStream ID, as specified in [I-D.ietf-mmusic-msid], Section 3. The track ID MUST be selected as described above.

o If the RtpTransceiver has a sendrecv or sendonly direction, and the application has specified RID values or has specified more than one encoding in the RtpSenders's parameters, an "a=rid" line for each encoding specified. The "a=rid" line is specified in [I-D.ietf-mmusic-rid], and its direction MUST be "send". If the application has chosen a RID value, it MUST be used as the rid-identifier; otherwise a RID value MUST be generated by the implementation. RID values MUST be generated in a fashion that does not leak user information, e.g., randomly or using a per-PeerConnection counter, and SHOULD be 3 bytes or less, to allow them to efficiently fit into the RTP header extension defined in [I-D.ietf-avtext-rid], Section 3. If no encodings have been specified, or only one encoding is specified but without a RID value, then no "a=rid" lines are generated.

o If the RtpTransceiver has a sendrecv or sendonly direction and more than one "a=rid" line has been generated, an "a=simulcast" line, with direction "send", as defined in [I-D.ietf-mmusic-sdp-simulcast], Section 6.2. The list of RIDs MUST include all of the RID identifiers used in the "a=rid" lines for this m= section.

o If the bundle policy for this PeerConnection is set to "max-bundle", and this is not the first m= section, or the bundle policy is set to "balanced", and this is not the first m= section for this media type, an "a=bundle-only" line.

The following attributes, which are of category IDENTICAL or TRANSPORT, MUST appear only in "m=" sections which either have a unique address or which are associated with the bundle-tag. (In initial offers, this means those "m=" sections which do not contain an "a=bundle-only" attribute.)

o "a=ice-ufrag" and "a=ice-pwd" lines, as specified in [RFC5245], Section 15.4.

o An "a=fingerprint" line for each of the endpoint's certificates, as specified in [RFC4572], Section 5; the digest algorithm used for the fingerprint MUST match that used in the certificate signature.

o An "a=setup" line, as specified in [RFC4145], Section 4, and clarified for use in DTLS-SRTP scenarios in [RFC5763], Section 5. The role value in the offer MUST be "actpass".

o An "a=dtls-id" line, as specified in [I-D.ietf-mmusic-dtls-sdp] Section 5.2.

o An "a=rtcp" line, as specified in [RFC3605], Section 2.1, containing the dummy value "9 IN IP4 0.0.0.0", because no candidates have yet been gathered.

o An "a=rtcp-mux" line, as specified in [RFC5761], Section 5.1.3.

o If the RTP/RTCP multiplexing policy is "require", an "a=rtcp-mux-only" line, as specified in [I-D.ietf-mmusic-mux-exclusive], Section 4.

o An "a=rtcp-rsize" line, as specified in [RFC5506], Section 5.

Lastly, if a data channel has been created, a m= section MUST be generated for data. The <media> field MUST be set to "application" and the <proto> field MUST be set to "UDP/DTLS/SCTP" [I-D.ietf-mmusic-sctp-sdp]. The "fmt" value MUST be set to "webrtc-datachannel" as specified in [I-D.ietf-mmusic-sctp-sdp], Section 4.1.

Within the data m= section, an "a=mid" line MUST be generated and included as described above, along with an "a=sctp-port" line referencing the SCTP port number, as defined in [I-D.ietf-mmusic-sctp-sdp], Section 5.1, and, if appropriate, an "a=max-message-size" line, as defined in [I-D.ietf-mmusic-sctp-sdp], Section 6.1.

As discussed above, the following attributes of category IDENTICAL or TRANSPORT are included only if the data m= section either has a unique address or is associated with the bundle-tag (e.g., if it is the only m= section):

o "a=ice-ufrag"

o "a=ice-pwd"

o "a=fingerprint"

o "a=setup"

o "a=dtls-id"

Once all m= sections have been generated, a session-level "a=group" attribute MUST be added as specified in [RFC5888]. This attribute MUST have semantics "BUNDLE", and MUST include the mid identifiers of each m= section. The effect of this is that the JSEP implementation offers all m= sections as one bundle group. However, whether the m= sections are bundle-only or not depends on the bundle policy.

The next step is to generate session-level lip sync groups as defined in [RFC5888], Section 7. For each MediaStream referenced by more than one RtpTransceiver (by passing those MediaStreams as arguments to the addTrack and addTransceiver methods), a group of type "LS" MUST be added that contains the mid values for each RtpTransceiver.

Attributes which SDP permits to either be at the session level or the media level SHOULD generally be at the media level even if they are identical. This promotes readability, especially if one of a set of initially identical attributes is subsequently changed.

Attributes other than the ones specified above MAY be included, except for the following attributes which are specifically incompatible with the requirements of [I-D.ietf-rtcweb-rtp-usage], and MUST NOT be included:

o "a=crypto"

o "a=key-mgmt"

o "a=ice-lite"

Note that when bundle is used, any additional attributes that are added MUST follow the advice in [I-D.ietf-mmusic-sdp-mux-attributes] on how those attributes interact with bundle.

Note that these requirements are in some cases stricter than those of SDP. Implementations MUST be prepared to accept compliant SDP even if it would not conform to the requirements for generating SDP in this specification.

5.2.2. Subsequent Offers

When createOffer is called a second (or later) time, or is called after a local description has already been installed, the processing is somewhat different than for an initial offer.

If the initial offer was not applied using setLocalDescription, meaning the PeerConnection is still in the "stable" state, the steps for generating an initial offer should be followed, subject to the following restriction:

o The fields of the "o=" line MUST stay the same except for the <session-version> field, which MUST increment by one on each call to createOffer if the offer might differ from the output of the previous call to createOffer; implementations MAY opt to increment <session-version> on every call. The value of the generated <session-version> is independent of the <session-version> of the current local description; in particular, in the case where the current version is N, an offer is created and applied with version N+1, and then that offer is rolled back so that the current version is again N, the next generated offer will still have version N+2.

Note that if the application creates an offer by reading currentLocalDescription instead of calling createOffer, the returned SDP may be different than when setLocalDescription was originally called, due to the addition of gathered ICE candidates, but the <session-version> will not have changed. There are no known scenarios in which this causes problems, but if this is a concern, the solution is simply to use createOffer to ensure a unique <session-version>.

If the initial offer was applied using setLocalDescription, but an answer from the remote side has not yet been applied, meaning the PeerConnection is still in the "local-offer" state, an offer is generated by following the steps in the "stable" state above, along with these exceptions:

o The "s=" and "t=" lines MUST stay the same.

o If any RtpTransceiver has been added, and there exists an m= section with a zero port in the current local description or the current remote description, that m= section MUST be recycled by generating an m= section for the added RtpTransceiver as if the m= section were being added to the session description, placed at the same index as the m= section with a zero port.

o If an RtpTransceiver is stopped and is not associated with an m= section, an m= section MUST NOT be generated for it. This prevents adding back RtpTransceivers whose m= sections were recycled and used for a new RtpTransceiver in a previous offer/answer exchange, as described above.

o If an RtpTransceiver has been stopped and is associated with an m= section, and the m= section is not being recycled as described above, an m= section MUST be generated for it with the port set to zero and all "a=msid" lines removed.

o For RtpTransceivers that are not stopped, the "a=msid" line(s) MUST stay the same if they are present in the current description, regardless of changes to the transceiver's direction or track. If no "a=msid" line is present in the current description, "a=msid" line(s) MUST be generated according to the same rules as for an initial offer.

o Each "m=" and c=" line MUST be filled in with the port, protocol, and address of the default candidate for the m= section, as described in [RFC5245], Section 4.3. If ICE checking has already completed for one or more candidate pairs and a candidate pair is in active use, then that pair MUST be used, even if ICE has not yet completed. Note that this differs from the guidance in [RFC5245], Section 9.1.2.2, which only refers to offers created when ICE has completed. In each case, if no RTP candidates have yet been gathered, dummy values MUST be used, as described above.

o Each "a=mid" line MUST stay the same.

o Each "a=ice-ufrag" and "a=ice-pwd" line MUST stay the same, unless the ICE configuration has changed (either changes to the supported STUN/TURN servers, or the ICE candidate policy), or the "IceRestart" option (Section 5.2.3.1 was specified. If the m= section is bundled into another m= section, it still MUST NOT contain any ICE credentials.

o If the m= section is not bundled into another m= section, its "a=rtcp" attribute line MUST be filled in with the port and address of the default RTCP candidate, as indicated in [RFC5761], Section 5.1.3. If no RTCP candidates have yet been gathered, dummy values MUST be used, as described in the initial offer section above.

o If the m= section is not bundled into another m= section, for each candidate that has been gathered during the most recent gathering phase (see Section 3.5.1), an "a=candidate" line MUST be added, as defined in [RFC5245], Section 4.3., paragraph 3. If candidate

gathering for the section has completed, an "a=end-of-candidates" attribute MUST be added, as described in [I-D.ietf-ice-trickle], Section 9.3. If the m= section is bundled into another m= section, both "a=candidate" and "a=end-of-candidates" MUST be omitted.

- o For RtpTransceivers that are still present, the "a=rid" lines MUST stay the same.

- o For RtpTransceivers that are still present, any "a=simulcast" line MUST stay the same.

- o If any RtpTransceiver has been stopped, the port MUST be set to zero and all "a=msid" lines MUST be removed.

- o If any RtpTransceiver has been added, and there exists a m= section with a zero port in the current local description or the current remote description, that m= section MUST be recycled by generating a m= section for the added RtpTransceiver as if the m= section were being added to session description, except that instead of adding it, the generated m= section replaces the m= section with a zero port. The new m= section MUST contain a new MID.

If the initial offer was applied using setLocalDescription, and an answer from the remote side has been applied using setRemoteDescription, meaning the PeerConnection is in the "remote-pranswer" or "stable" states, an offer is generated based on the negotiated session descriptions by following the steps mentioned for the "local-offer" state above.

In addition, for each non-recycled, non-rejected m= section in the new offer, the following adjustments are made based on the contents of the corresponding m= section in the current remote description, if any:

- o The m= line and corresponding "a=rtpmap" and "a=fmtp" lines MUST only include codecs present in the most recent answer which have not been excluded by the codec preferences of the associated transceiver. Note that non-JSEP endpoints are not subject to these restrictions, and might offer media formats that were not present in the most recent answer, as specified in [RFC3264], Section 8. Therefore, JSEP implementations MUST be prepared to receive such offers.

- o Unless codec preferences have been set for the associated transceiver, the media formats on the m= line MUST be generated in the same order as in the current local description.

- o The RTP header extensions MUST only include those that are present in the most recent answer.

- o The RTCP feedback extensions MUST only include those that are present in the most recent answer.

- o The "a=rtcp" line MUST only be added if the most recent answer did not include an "a=rtcp-mux" line.

- o The "a=rtcp-mux" line MUST only be added if present in the most recent answer.

- o The "a=rtcp-mux-only" line MUST NOT be added.

- o The "a=rtcp-rsize" line MUST only be added if present in the most recent answer.

- o An "a=bundle-only" line MUST NOT be added, as indicated in [I-D.ietf-mmusic-sdp-bundle-negotiation], Section 6. Instead, JSEP implementations MUST simply omit parameters in the IDENTICAL and TRANSPORT categories for bundled m= sections, as described in [I-D.ietf-mmusic-sdp-bundle-negotiation], Section 8.1.

- o Note that if media m= sections are bundled into a data m= section, then certain TRANSPORT and IDENTICAL attributes may appear in the data m= section even if they would otherwise only be appropriate for a media m= section (e.g., "a=rtcp-mux"). This cannot happen in initial offers because in the initial offer JSEP implementations always list media m= sections (if any) before the data m= section (if any), and at least one of those media m= sections will not have the "a=bundle-only" attribute. Therefore, in initial offers, any "a=bundle-only" m= sections will be bundled into a preceding non-bundle-only media m= section.

The "a=group:BUNDLE" attribute MUST include the MID identifiers specified in the bundle group in the most recent answer, minus any m= sections that have been marked as rejected, plus any newly added or re-enabled m= sections. In other words, the bundle attribute must contain all m= sections that were previously bundled, as long as they are still alive, as well as any new m= sections.

"a=group:LS" attributes are generated in the same way as for initial offers, with the additional stipulation that any lip sync groups that were present in the most recent answer MUST continue to exist and MUST contain any previously existing MID identifiers, as long as the identified m= sections still exist and are not rejected, and the group still contains at least two MID identifiers. This ensures that

any synchronized "recvonly" m= sections continue to be synchronized in the new offer.

5.2.3. Options Handling

The createOffer method takes as a parameter an RTCOfferOptions object. Special processing is performed when generating a SDP description if the following options are present.

5.2.3.1. IceRestart

If the "IceRestart" option is specified, with a value of "true", the offer MUST indicate an ICE restart by generating new ICE ufrag and pwd attributes, as specified in [RFC5245], Section 9.1.1.1. If this option is specified on an initial offer, it has no effect (since a new ICE ufrag and pwd are already generated). Similarly, if the ICE configuration has changed, this option has no effect, since new ufrag and pwd attributes will be generated automatically. This option is primarily useful for reestablishing connectivity in cases where failures are detected by the application.

5.2.3.2. VoiceActivityDetection

If the "VoiceActivityDetection" option is specified, with a value of "true", the offer MUST indicate support for silence suppression in the audio it receives by including comfort noise ("CN") codecs for each offered audio codec, as specified in [RFC3389], Section 5.1, except for codecs that have their own internal silence suppression support. For codecs that have their own internal silence suppression support, the appropriate fmtp parameters for that codec MUST be specified to indicate that silence suppression for received audio is desired. For example, when using the Opus codec [RFC6716], the "usedtx=1" parameter, specified in [RFC7587], would be used in the offer. This option allows the endpoint to significantly reduce the amount of audio bandwidth it receives, at the cost of some fidelity, depending on the quality of the remote VAD algorithm.

If the "VoiceActivityDetection" option is specified, with a value of "false", the JSEP implementation MUST NOT emit "CN" codecs. For codecs that have their own internal silence suppression support, the appropriate fmtp parameters for that codec MUST be specified to indicate that silence suppression for received audio is not desired. For example, when using the Opus codec, the "usedtx=0" parameter would be specified in the offer.

Note that setting the "VoiceActivityDetection" parameter when generating an offer is a request to receive audio with silence

suppression. It has no impact on whether the local endpoint does silence suppression for the audio it sends.

The "VoiceActivityDetection" option does not have any impact on the setting of the "vad" value in the signaling of the client to mixer audio level header extension described in [RFC6464], Section 4.

5.3. Generating an Answer

When createAnswer is called, a new SDP description must be created that is compatible with the supplied remote description as well as the requirements specified in [I-D.ietf-rtcweb-rtp-usage]. The exact details of this process are explained below.

5.3.1. Initial Answers

When createAnswer is called for the first time after a remote description has been provided, the result is known as the initial answer. If no remote description has been installed, an answer cannot be generated, and an error MUST be returned.

Note that the remote description SDP may not have been created by a JSEP endpoint and may not conform to all the requirements listed in Section 5.2. For many cases, this is not a problem. However, if any mandatory SDP attributes are missing, or functionality listed as mandatory-to-use above is not present, this MUST be treated as an error, and MUST cause the affected m= sections to be marked as rejected.

The first step in generating an initial answer is to generate session-level attributes. The process here is identical to that indicated in the initial offers section above, except that the "a=ice-options" line, with the "trickle" option as specified in [I-D.ietf-ice-trickle], Section 4, is only included if such an option was present in the offer.

The next step is to generate session-level lip sync groups, as defined in [RFC5888], Section 7. For each group of type "LS" present in the offer, select the local RtpTransceivers that are referenced by the MID values in the specified group, and determine which of them either reference a common local MediaStream (specified in the calls to addTrack/addTransceiver used to create them), or have no MediaStream to reference because they were not created by addTrack/addTransceiver. If at least two such RtpTransceivers exist, a group of type "LS" with the mid values of these RtpTransceivers MUST be added. Otherwise the offered "LS" group MUST be ignored and no corresponding group generated in the answer.

As a simple example, consider the following offer of a single audio and single video track contained in the same MediaStream. SDP lines not relevant to this example have been removed for clarity. As explained in Section 5.2, a group of type "LS" has been added that references each track's RtpTransceiver.

```
a=group:LS a1 v1
m=audio 10000 UDP/TLS/RTP/SAVPF 0
a=mid:a1
a=msid:ms1 mst1a
m=video 10001 UDP/TLS/RTP/SAVPF 96
a=mid:v1
a=msid:ms1 mst1v
```

If the answerer uses a single MediaStream when it adds its tracks, both of its transceivers will reference this stream, and so the subsequent answer will contain a "LS" group identical to that in the offer, as shown below:

```
a=group:LS a1 v1
m=audio 20000 UDP/TLS/RTP/SAVPF 0
a=mid:a1
a=msid:ms2 mst2a
m=video 20001 UDP/TLS/RTP/SAVPF 96
a=mid:v1
a=msid:ms2 mst2v
```

However, if the answerer groups its tracks into separate MediaStreams, its transceivers will reference different streams, and so the subsequent answer will not contain a "LS" group.

```
m=audio 20000 UDP/TLS/RTP/SAVPF 0
a=mid:a1
a=msid:ms2a mst2a
m=video 20001 UDP/TLS/RTP/SAVPF 96
a=mid:v1
a=msid:ms2b mst2v
```

Finally, if the answerer does not add any tracks, its transceivers will not reference any MediaStreams, causing the preferences of the offerer to be maintained, and so the subsequent answer will contain an identical "LS" group.

```
a=group:LS a1 v1
m=audio 20000 UDP/TLS/RTP/SAVPF 0
a=mid:a1
a=recvonly
m=video 20001 UDP/TLS/RTP/SAVPF 96
a=mid:v1
a=recvonly
```

The Section 7.2 example later in this document shows a more involved case of "LS" group generation.

The next step is to generate m= sections for each m= section that is present in the remote offer, as specified in [RFC3264], Section 6. For the purposes of this discussion, any session-level attributes in the offer that are also valid as media-level attributes are considered to be present in each m= section.

The next step is to go through each offered m= section. Each offered m= section will have an associated RtpTransceiver, as described in Section 5.9. If there are more RtpTransceivers than there are m= sections, the unmatched RtpTransceivers will need to be associated in a subsequent offer.

For each offered m= section, if any of the following conditions are true, the corresponding m= section in the answer MUST be marked as rejected by setting the port in the m= line to zero, as indicated in [RFC3264], Section 6., and further processing for this m= section can be skipped:

- o The associated RtpTransceiver has been stopped.

- o No supported codec is present in the offer.

- o The bundle policy is "max-bundle", and this is not the first m= section or in the same bundle group as the first m= section.

- o The bundle policy is "balanced", and this is not the first m= section for this media type or in the same bundle group as the first m= section for this media type.

Otherwise, each m= section in the answer should then be generated as specified in [RFC3264], Section 6.1. For the m= line itself, the following rules must be followed:

- o The port value would normally be set to the port of the default ICE candidate for this m= section, but given that no candidates

are available yet, the "dummy" port value of 9 (Discard) MUST be used, as indicated in [I-D.ietf-ice-trickle], Section 5.1.

- The <proto> field MUST be set to exactly match the <proto> field for the corresponding m= line in the offer.

- If codec preferences have been set for the associated transceiver, media formats MUST be generated in the corresponding order, and MUST exclude any codecs not present in the codec preferences or not present in the offer. Note that non-JSEP endpoints are not subject to this restriction, and might add media formats in the answer that are not present in the offer, as specified in [RFC3264], Section 6.1. Therefore, JSEP implementations MUST be prepared to receive such answers.

- Unless excluded by the above restrictions, the media formats MUST include the mandatory audio/video codecs as specified in [I-D.ietf-rtcweb-audio](see Section 3) and [I-D.ietf-rtcweb-video](see Section 5).

The m= line MUST be followed immediately by a "c=" line, as specified in [RFC4566], Section 5.7. Again, as no candidates are available yet, the "c=" line must contain the "dummy" value "IN IP4 0.0.0.0", as defined in [I-D.ietf-ice-trickle], Section 5.1.

If the offer supports bundle, all m= sections to be bundled must use the same ICE credentials and candidates; all m= sections not being bundled must use unique ICE credentials and candidates. Each m= section MUST contain the following attributes (which are of attribute types other than IDENTICAL and TRANSPORT):

- If and only if present in the offer, an "a=mid" line, as specified in [RFC5888], Section 9.1. The "mid" value MUST match that specified in the offer.

- A direction attribute, determined by applying the rules regarding the offered direction specified in [RFC3264], Section 6.1, and then intersecting with the direction of the associated RtpTransceiver. For example, in the case where an m= section is offered as "sendonly", and the local transceiver is set to "sendrecv", the result in the answer is a "recvonly" direction.

- For each media format on the m= line, "a=rtpmap" and "a=fmtp" lines, as specified in [RFC4566], Section 6, and [RFC3264], Section 6.1.

- If "rtx" is present in the offer, for each primary codec where RTP retransmission should be used, a corresponding "a=rtpmap" line

indicating "rtx" with the clock rate of the primary codec and an "a=fmtp" line that references the payload type of the primary codec, as specified in [RFC4588], Section 8.1.

o For each supported FEC mechanism, "a=rtpmap" and "a=fmtp" lines, as specified in [RFC4566], Section 6. The FEC mechanisms that MUST be supported are specified in [I-D.ietf-rtcweb-fec], Section 6, and specific usage for each media type is outlined in Sections 4 and 5.

o If this m= section is for media with configurable durations of media per packet, e.g., audio, an "a=maxptime" line, as described in Section 5.2.

o If this m= section is for video media, and there are known limitations on the size of images which can be decoded, an "a=imageattr" line, as specified in Section 3.6.

o For each supported RTP header extension that is present in the offer, an "a=extmap" line, as specified in [RFC5285], Section 5. The list of header extensions that SHOULD/MUST be supported is specified in [I-D.ietf-rtcweb-rtp-usage], Section 5.2. Any header extensions that require encryption MUST be specified as indicated in [RFC6904], Section 4.

o For each supported RTCP feedback mechanism that is present in the offer, an "a=rtcp-fb" mechanism, as specified in [RFC4585], Section 4.2. The list of RTCP feedback mechanisms that SHOULD/MUST be supported is specified in [I-D.ietf-rtcweb-rtp-usage], Section 5.1.

o If the RtpTransceiver has a sendrecv or sendonly direction:

  * For each MediaStream that was associated with the transceiver when it was created via addTrack or addTransceiver, an "a=msid" line, as specified in [I-D.ietf-mmusic-msid], Section 2. If a MediaStreamTrack is attached to the transceiver's RtpSender, the "a=msid" lines MUST use that track's ID. If no MediaStreamTrack is attached, a valid ID MUST be generated, in the same way that the implementation generates IDs for local tracks.

  * If no MediaStream is associated with the transceiver, a single "a=msid" line with the special value "-" in place of the MediaStream ID, as specified in [I-D.ietf-mmusic-msid], Section 3. The track ID MUST be selected as described above.

Each m= section which is not bundled into another m= section, MUST contain the following attributes (which are of category IDENTICAL or TRANSPORT):

o "a=ice-ufrag" and "a=ice-pwd" lines, as specified in [RFC5245], Section 15.4.

o An "a=fingerprint" line for each of the endpoint's certificates, as specified in [RFC4572], Section 5; the digest algorithm used for the fingerprint MUST match that used in the certificate signature.

o An "a=setup" line, as specified in [RFC4145], Section 4, and clarified for use in DTLS-SRTP scenarios in [RFC5763], Section 5. The role value in the answer MUST be "active" or "passive"; the "active" role is RECOMMENDED.

o An "a=dtls-id" line, as specified in [I-D.ietf-mmusic-dtls-sdp] Section 5.3.

o If present in the offer, an "a=rtcp-mux" line, as specified in [RFC5761], Section 5.1.3. Otherwise, an "a=rtcp" line, as specified in [RFC3605], Section 2.1, containing the dummy value "9 IN IP4 0.0.0.0" (because no candidates have yet been gathered).

o If present in the offer, an "a=rtcp-rsize" line, as specified in [RFC5506], Section 5.

If a data channel m= section has been offered, a m= section MUST also be generated for data. The <media> field MUST be set to "application" and the <proto> and <fmt> fields MUST be set to exactly match the fields in the offer.

Within the data m= section, an "a=mid" line MUST be generated and included as described above, along with an "a=sctp-port" line referencing the SCTP port number, as defined in [I-D.ietf-mmusic-sctp-sdp], Section 5.1, and, if appropriate, an "a=max-message-size" line, as defined in [I-D.ietf-mmusic-sctp-sdp], Section 6.1.

As discussed above, the following attributes of category IDENTICAL or TRANSPORT are included only if the data m= section is not bundled into another m= section:

o "a=ice-ufrag"

o "a=ice-pwd"

o "a=fingerprint"

o "a=setup"

o "a=dtls-id"

Note that if media m= sections are bundled into a data m= section, then certain TRANSPORT and IDENTICAL attributes may also appear in the data m= section even if they would otherwise only be appropriate for a media m= section (e.g., "a=rtcp-mux").

If "a=group" attributes with semantics of "BUNDLE" are offered, corresponding session-level "a=group" attributes MUST be added as specified in [RFC5888]. These attributes MUST have semantics "BUNDLE", and MUST include the all mid identifiers from the offered bundle groups that have not been rejected. Note that regardless of the presence of "a=bundle-only" in the offer, no m= sections in the answer should have an "a=bundle-only" line.

Attributes that are common between all m= sections MAY be moved to session-level, if explicitly defined to be valid at session-level.

The attributes prohibited in the creation of offers are also prohibited in the creation of answers.

5.3.2. Subsequent Answers

When createAnswer is called a second (or later) time, or is called after a local description has already been installed, the processing is somewhat different than for an initial answer.

If the initial answer was not applied using setLocalDescription, meaning the PeerConnection is still in the "have-remote-offer" state, the steps for generating an initial answer should be followed, subject to the following restriction:

o The fields of the "o=" line MUST stay the same except for the <session-version> field, which MUST increment if the session description changes in any way from the previously generated answer.

If any session description was previously supplied to setLocalDescription, an answer is generated by following the steps in the "have-remote-offer" state above, along with these exceptions:

o The "s=" and "t=" lines MUST stay the same.

o Each "m=" and c=" line MUST be filled in with the port and address of the default candidate for the m= section, as described in [RFC5245], Section 4.3. Note, however, that the m= line protocol need not match the default candidate, because this protocol value must instead match what was supplied in the offer, as described above.

o Unless codec preferences have been set for the associated transceiver, the media formats on the m= line MUST be generated in the same order as in the current local description.

o Each "a=ice-ufrag" and "a=ice-pwd" line MUST stay the same, unless the m= section is restarting, in which case new ICE credentials must be created as specified in [RFC5245], Section 9.2.1.1. If the m= section is bundled into another m= section, it still MUST NOT contain any ICE credentials.

o Each "a=setup" line MUST use an "active" or "passive" role value consistent with the existing DTLS association, if the association is being continued by the offerer.

o If the m= section is not bundled into another m= section and RTCP multiplexing is not active, an "a=rtcp" attribute line MUST be filled in with the port and address of the default RTCP candidate. If no RTCP candidates have yet been gathered, dummy values MUST be used, as described in the initial answer section above.

o If the m= section is not bundled into another m= section, for each candidate that has been gathered during the most recent gathering phase (see Section 3.5.1), an "a=candidate" line MUST be added, as defined in [RFC5245], Section 4.3., paragraph 3. If candidate gathering for the section has completed, an "a=end-of-candidates" attribute MUST be added, as described in [I-D.ietf-ice-trickle], Section 9.3. If the m= section is bundled into another m= section, both "a=candidate" and "a=end-of-candidates" MUST be omitted.

o For RtpTransceivers that are not stopped, the "a=msid" line(s) MUST stay the same, regardless of changes to the transceiver's direction or track. If no "a=msid" line is present in the current description, "a=msid" line(s) MUST be generated according to the same rules as for an initial answer.

5.3.3. Options Handling

The createAnswer method takes as a parameter an RTCAnswerOptions object. The set of parameters for RTCAnswerOptions is different than those supported in RTCOfferOptions; the IceRestart option is

unnecessary, as ICE credentials will automatically be changed for all m= sections where the offerer chose to perform ICE restart.

The following options are supported in RTCAnswerOptions.

5.3.3.1. VoiceActivityDetection

Silence suppression in the answer is handled as described in Section 5.2.3.2, with one exception: if support for silence suppression was not indicated in the offer, the VoiceActivityDetection parameter has no effect, and the answer should be generated as if VoiceActivityDetection was set to false. This is done on a per-codec basis (e.g., if the offerer somehow offered support for CN but set "usedtx=0" for Opus, setting VoiceActivityDetection to true would result in an answer with CN codecs and "usedtx=0").

5.4. Modifying an Offer or Answer

The SDP returned from createOffer or createAnswer MUST NOT be changed before passing it to setLocalDescription. If precise control over the SDP is needed, the aforementioned createOffer/createAnswer options or RtpTransceiver APIs MUST be used.

Note that the application MAY modify the SDP to reduce the capabilities in the offer it sends to the far side (post-setLocalDescription) or the offer that it installs from the far side (pre-setRemoteDescription), as long as it remains a valid SDP offer and specifies a subset of what was in the original offer. This is safe because the answer is not permitted to expand capabilities, and therefore will just respond to what is present in the offer.

The application SHOULD NOT modify the SDP in the answer it transmits, as the answer contains the negotiated capabilities, and this can cause the two sides to have different ideas about what exactly was negotiated.

As always, the application is solely responsible for what it sends to the other party, and all incoming SDP will be processed by the JSEP implementation to the extent of its capabilities. It is an error to assume that all SDP is well-formed; however, one should be able to assume that any implementation of this specification will be able to process, as a remote offer or answer, unmodified SDP coming from any other implementation of this specification.

5.5. Processing a Local Description

When a SessionDescription is supplied to setLocalDescription, the following steps MUST be performed:

- First, the type of the SessionDescription is checked against the current state of the PeerConnection:

  * If the type is "offer", the PeerConnection state MUST be either "stable" or "have-local-offer".

  * If the type is "pranswer" or "answer", the PeerConnection state MUST be either "have-remote-offer" or "have-local-pranswer".

- If the type is not correct for the current state, processing MUST stop and an error MUST be returned.

- The SessionDescription is then checked to ensure that its contents are identical to those generated in the last call to createOffer/createAnswer, and thus have not been altered, as discussed in Section 5.4; otherwise, processing MUST stop and an error MUST be returned.

- Next, the SessionDescription is parsed into a data structure, as described in the Section 5.7 section below. If parsing fails for any reason, processing MUST stop and an error MUST be returned.

- Finally, the parsed SessionDescription is applied as described in the Section 5.8 section below.

5.6. Processing a Remote Description

When a SessionDescription is supplied to setRemoteDescription, the following steps MUST be performed:

- First, the type of the SessionDescription is checked against the current state of the PeerConnection:

  * If the type is "offer", the PeerConnection state MUST be either "stable" or "have-remote-offer".

  * If the type is "pranswer" or "answer", the PeerConnection state MUST be either "have-local-offer" or "have-remote-pranswer".

- If the type is not correct for the current state, processing MUST stop and an error MUST be returned.

o Next, the SessionDescription is parsed into a data structure, as described in the Section 5.7 section below. If parsing fails for any reason, processing MUST stop and an error MUST be returned.

o Finally, the parsed SessionDescription is applied as described in the Section 5.9 section below.

5.7. Parsing a Session Description

When a SessionDescription of any type is supplied to setLocal/RemoteDescription, the implementation must parse it and reject it if it is invalid. The exact details of this process are explained below.

The SDP contained in the session description object consists of a sequence of text lines, each containing a key-value expression, as described in [RFC4566], Section 5. The SDP is read, line-by-line, and converted to a data structure that contains the deserialized information. However, SDP allows many types of lines, not all of which are relevant to JSEP applications. For each line, the implementation will first ensure it is syntactically correct according to its defining ABNF, check that it conforms to [RFC4566] and [RFC3264] semantics, and then either parse and store or discard the provided value, as described below.

If any line is not well-formed, or cannot be parsed as described, the parser MUST stop with an error and reject the session description, even if the value is to be discarded. This ensures that implementations do not accidentally misinterpret ambiguous SDP.

5.7.1. Session-Level Parsing

First, the session-level lines are checked and parsed. These lines MUST occur in a specific order, and with a specific syntax, as defined in [RFC4566], Section 5. Note that while the specific line types (e.g. "v=", "c=") MUST occur in the defined order, lines of the same type (typically "a=") can occur in any order, and their ordering is not meaningful.

The following non-attribute lines are not meaningful in the JSEP context and MAY be discarded once they have been checked.

The "c=" line MUST be checked for syntax but its value is not used. This supersedes the guidance in [RFC5245], Section 6.1, to use "ice-mismatch" to indicate mismatches between "c=" and the candidate lines; because JSEP always uses ICE, "ice-mismatch" is not useful in this context.

The "i=", "u=", "e=", "p=", "t=", "r=", "z=", and "k=" lines are not used by this specification; they MUST be checked for syntax but their values are not used.

The remaining non-attribute lines are processed as follows:

The "v=" line MUST have a version of 0, as specified in [RFC4566], Section 5.1.

The "o=" line MUST be parsed as specified in [RFC4566], Section 5.2.

The "b=" line, if present, MUST be parsed as specified in [RFC4566], Section 5.8, and the bwtype and bandwidth values stored.

Finally, the attribute lines are processed. Specific processing MUST be applied for the following session-level attribute ("a=") lines:

- Any "a=group" lines are parsed as specified in [RFC5888], Section 5, and the group's semantics and mids are stored.

- If present, a single "a=ice-lite" line is parsed as specified in [RFC5245], Section 15.3, and a value indicating the presence of ice-lite is stored.

- If present, a single "a=ice-ufrag" line is parsed as specified in [RFC5245], Section 15.4, and the ufrag value is stored.

- If present, a single "a=ice-pwd" line is parsed as specified in [RFC5245], Section 15.4, and the password value is stored.

- If present, a single "a=ice-options" line is parsed as specified in [RFC5245], Section 15.5, and the set of specified options is stored.

- Any "a=fingerprint" lines are parsed as specified in [RFC4572], Section 5, and the set of fingerprint and algorithm values is stored.

- If present, a single "a=setup" line is parsed as specified in [RFC4145], Section 4, and the setup value is stored.

- If present, a single "a=dtls-id" line is parsed as specified in [I-D.ietf-mmusic-dtls-sdp] Section 5, and the dtls-id value is stored.

- o Any "a=identity" lines are parsed and the identity values stored for subsequent verification, as specified [I-D.ietf-rtcweb-security-arch], Section 5.

- o Any "a=extmap" lines are parsed as specified in [RFC5285], Section 5, and their values are stored.

As required by [RFC4566], Section 5.13, unknown attribute lines MUST be ignored.

Once all the session-level lines have been parsed, processing continues with the lines in m= sections.

5.7.2. Media Section Parsing

Like the session-level lines, the media section lines MUST occur in the specific order and with the specific syntax defined in [RFC4566], Section 5.

The "m=" line itself MUST be parsed as described in [RFC4566], Section 5.14, and the media, port, proto, and fmt values stored.

Following the "m=" line, specific processing MUST be applied for the following non-attribute lines:

- o As with the "c=" line at the session level, the "c=" line MUST be parsed according to [RFC4566], Section 5.7, but its value is not used.

- o The "b=" line, if present, MUST be parsed as specified in [RFC4566], Section 5.8, and the bwtype and bandwidth values stored.

Specific processing MUST also be applied for the following attribute lines:

- o If present, a single "a=ice-ufrag" line is parsed as specified in [RFC5245], Section 15.4, and the ufrag value is stored.

- o If present, a single "a=ice-pwd" line is parsed as specified in [RFC5245], Section 15.4, and the password value is stored.

- o If present, a single "a=ice-options" line is parsed as specified in [RFC5245], Section 15.5, and the set of specified options is stored.

- o Any "a=candidate" attributes MUST be parsed as specified in [RFC5245], Section 15.1, and their values stored.

o Any "a=remote-candidates" attributes MUST be parsed as specified in [RFC5245], Section 15.2, but their values are ignored.

o If present, a single "a=end-of-candidates" attribute MUST be parsed as specified in [I-D.ietf-ice-trickle], Section 8.2, and its presence or absence flagged and stored.

o Any "a=fingerprint" lines are parsed as specified in [RFC4572], Section 5, and the set of fingerprint and algorithm values is stored.

If the "m=" proto value indicates use of RTP, as described in the Section 5.1.2 section above, the following attribute lines MUST be processed:

o The "m=" fmt value MUST be parsed as specified in [RFC4566], Section 5.14, and the individual values stored.

o Any "a=rtpmap" or "a=fmtp" lines MUST be parsed as specified in [RFC4566], Section 6, and their values stored.

o If present, a single "a=ptime" line MUST be parsed as described in [RFC4566], Section 6, and its value stored.

o If present, a single "a=maxptime" line MUST be parsed as described in [RFC4566], Section 6, and its value stored.

o If present, a single direction attribute line (e.g. "a=sendrecv") MUST be parsed as described in [RFC4566], Section 6, and its value stored.

o Any "a=ssrc" or "a=ssrc-group" attributes MUST be parsed as specified in [RFC5576], Sections 4.1-4.2, and their values stored.

o Any "a=extmap" attributes MUST be parsed as specified in [RFC5285], Section 5, and their values stored.

o Any "a=rtcp-fb" attributes MUST be parsed as specified in [RFC4585], Section 4.2., and their values stored.

o If present, a single "a=rtcp-mux" attribute MUST be parsed as specified in [RFC5761], Section 5.1.3, and its presence or absence flagged and stored.

o If present, a single "a=rtcp-mux-only" attribute MUST be parsed as specified in [I-D.ietf-mmusic-mux-exclusive], Section 3, and its presence or absence flagged and stored.

- o If present, a single "a=rtcp-rsize" attribute MUST be parsed as specified in [RFC5506], Section 5, and its presence or absence flagged and stored.

- o If present, a single "a=rtcp" attribute MUST be parsed as specified in [RFC3605], Section 2.1, but its value is ignored, as this information is superfluous when using ICE.

- o If present, "a=msid" attributes MUST be parsed as specified in [I-D.ietf-mmusic-msid], Section 3.2, and their values stored.

- o Any "a=imageattr" attributes MUST be parsed as specified in [RFC6236], Section 3, and their values stored.

- o Any "a=rid" lines MUST be parsed as specified in [I-D.ietf-mmusic-rid], Section 10, and their values stored.

- o If present, a single "a=simulcast" line MUST be parsed as specified in [I-D.ietf-mmusic-sdp-simulcast], and its values stored.

Otherwise, if the "m=" proto value indicates use of SCTP, the following attribute lines MUST be processed:

- o The "m=" fmt value MUST be parsed as specified in [I-D.ietf-mmusic-sctp-sdp], Section 4.3, and the application protocol value stored.

- o An "a=sctp-port" attribute MUST be present, and it MUST be parsed as specified in [I-D.ietf-mmusic-sctp-sdp], Section 5.2, and the value stored.

- o If present, a single "a=max-message-size" attribute MUST be parsed as specified in [I-D.ietf-mmusic-sctp-sdp], Section 6, and the value stored. Otherwise, use the specified default.

As required by [RFC4566], Section 5.13, unknown attribute lines MUST be ignored.

5.7.3. Semantics Verification

Assuming parsing completes successfully, the parsed description is then evaluated to ensure internal consistency as well as proper support for mandatory features. Specifically, the following checks are performed:

- o For each m= section, valid values for each of the mandatory-to-use features enumerated in Section 5.1.1 MUST be present. These

values MAY either be present at the media level, or inherited from the session level.

* ICE ufrag and password values, which MUST comply with the size limits specified in [RFC5245], Section 15.4.

* dtls-id value, which MUST be set according to [I-D.ietf-mmusic-dtls-sdp] Section 5. If this is a re-offer and the dtls-id value is different from that presently in use, the DTLS connection is not being continued and the remote description MUST be part of an ICE restart, together with new ufrag and password values. If this is an answer, the dtls-id value, if present, MUST be the same as in the offer.

* DTLS setup value, which MUST be set according to the rules specified in [RFC5763], Section 5 and MUST be consistent with the selected role of the current DTLS connection, if one exists and is being continued.

* DTLS fingerprint values, where at least one fingerprint MUST be present.

o All RID values referenced in an "a=simulcast" line MUST exist as "a=rid" lines.

o Each m= section is also checked to ensure prohibited features are not used. If this is a local description, the "ice-lite" attribute MUST NOT be specified.

o If the RTP/RTCP multiplexing policy is "require", each m= section MUST contain an "a=rtcp-mux" attribute. If an "m=" section contains an "a=rtcp-mux-only" attribute then that section MUST also contain an "a=rtcp-mux" attribute.

If this session description is of type "pranswer" or "answer", the following additional checks are applied:

o The session description must follow the rules defined in [RFC3264], Section 6, including the requirement that the number of m= sections MUST exactly match the number of m= sections in the associated offer.

o For each m= section, the media type and protocol values MUST exactly match the media type and protocol values in the corresponding m= section in the associated offer.

If any of the preceding checks failed, processing MUST stop and an error MUST be returned.

5.8. Applying a Local Description

The following steps are performed at the media engine level to apply a local description. If an error is returned, the session MUST be restored to the state it was in before performing these steps.

Next, m= sections are processed. For each m= section, the following steps MUST be performed; if any parameters are out of bounds, or cannot be applied, processing MUST stop and an error MUST be returned.

o If this m= section is new, begin gathering candidates for it, as defined in [RFC5245], Section 4.1.1, unless it has been marked as bundle-only.

o Or, if the ICE ufrag and password values have changed, and it has not been marked as bundle-only, trigger the ICE agent to start an ICE restart, and begin gathering new candidates for the m= section as described in [RFC5245], Section 9.1.1.1. If this description is an answer, also start checks on that media section as defined in [RFC5245], Section 9.3.1.1.

o If the m= section proto value indicates use of RTP:

  * If there is no RtpTransceiver associated with this m= section (which will only happen when applying an offer), find one and associate it with this m= section according to the following steps:

    + Find the RtpTransceiver that corresponds to this m= section, using the mapping between transceivers and m= section indices established when creating the offer.

    + Set the value of this RtpTransceiver's mid property to the MID of the m= section.

  * If RTCP mux is indicated, prepare to demux RTP and RTCP from the RTP ICE component, as specified in [RFC5761], Section 5.1.3. If RTCP mux is not indicated, but was previously negotiated, i.e., the RTCP ICE component no longer exists, this MUST result in an error.

  * For each specified RTP header extension, establish a mapping between the extension ID and URI, as described in section 6 of [RFC5285]. If any indicated RTP header extension is not supported, this MUST result in an error.

   * If the MID header extension is supported, prepare to demux RTP streams intended for this m= section based on the MID header extension, as described in [I-D.ietf-mmusic-sdp-bundle-negotiation], Section 14.

   * For each specified media format, establish a mapping between the payload type and the actual media format, as described in [RFC3264], Section 6.1. If any indicated media format is not supported, this MUST result in an error.

   * For each specified "rtx" media format, establish a mapping between the RTX payload type and its associated primary payload type, as described in [RFC4588], Sections 8.6 and 8.7. If any referenced primary payload types are not present, this MUST result in an error.

   * If the directional attribute is of type "sendrecv" or "recvonly", enable receipt and decoding of media.

Finally, if this description is of type "pranswer" or "answer", follow the processing defined in the Section 5.10 section below.

5.9. Applying a Remote Description

The following steps are performed to apply a remote description. If an error is returned, the session MUST be restored to the state it was in before performing these steps.

If the answer contains any "a=ice-options" attributes where "trickle" is listed as an attribute, update the PeerConnection canTrickle property to be true. Otherwise, set this property to false.

The following steps MUST be performed for attributes at the session level; if any parameters are out of bounds, or cannot be applied, processing MUST stop and an error MUST be returned.

o For any specified "CT" bandwidth value, set this as the limit for the maximum total bitrate for all m= sections, as specified in Section 5.8 of [RFC4566]. Within this overall limit, the implementation can dynamically decide how to best allocate the available bandwidth between m= sections, respecting any specific limits that have been specified for individual m= sections.

o For any specified "RR" or "RS" bandwidth values, handle as specified in [RFC3556], Section 2.

o Any "AS" bandwidth value MUST be ignored, as the meaning of this construct at the session level is not well defined.

For each m= section, the following steps MUST be performed; if any parameters are out of bounds, or cannot be applied, processing MUST stop and an error MUST be returned.

o If the PeerConnection state is "have-local-offer", and the ICE ufrag or password changed from the previous remote description, then an ICE restart is needed, as described in Section 9.1.1.1 of [RFC5245]. If the description is of type "offer", note that an ICE restart is needed. If the description is of type "answer" or "pranswer" and the current local description is also an ICE restart, then signal the ICE agent to begin checks as described in Section 9.3.1.1 of [RFC5245]. An answerer MUST change the ufrag and password in an answer if and only if ICE is restarting, as described in Section 9.2.1.1 of [RFC5245].

o If the PeerConnection state is "have-remote-pranswer", and the ICE ufrag or password changed from the previous provisional answer, then signal the ICE agent to discard any previous ICE check list state for the m= section and begin checks as if this were the first answer. However, such an answer MAY only change the ICE ufrag or password if the local offer is starting or restarting ICE for the m= section.

o Configure the ICE components associated with this media section to use the supplied ICE remote ufrag and password for their connectivity checks.

o Pair any supplied ICE candidates with any gathered local candidates, as described in Section 5.7 of [RFC5245] and start connectivity checks with the appropriate credentials.

o If an "a=end-of-candidates" attribute is present, process the end-of-candidates indication as described in [I-D.ietf-ice-trickle] Section 11.

o If the m= section proto value indicates use of RTP:

   * If the m= section is being recycled (see Section 5.2.2), dissociate the currently associated RtpTransceiver by setting its mid property to null, and discard the mapping between the transceiver and its m= section index.

   * If the m= section is not associated with any RtpTransceiver (possibly because it was dissociated in the previous step), either find an RtpTransceiver or create one according to the following steps:

   + If the m= section is sendrecv or recvonly, and there are RtpTransceivers of the same type that were added to the PeerConnection by addTrack and are not associated with any m= section and are not stopped, find the first (according to the canonical order described in Section 5.2.1) such RtpTransceiver.

   + If no RtpTransceiver was found in the previous step, create one with a recvonly direction.

   + Associate the found or created RtpTransceiver with the m= section by setting the value of the RtpTransceiver's mid property to the MID of the m= section, and establish a mapping between the transceiver and the index of the m= section. If the m= section does not include a MID (i.e., the remote endpoint does not support the MID extension), generate a value for the RtpTransceiver mid property, following the guidance for "a=mid" mentioned in Section 5.2.1.

* For each specified media format that is also supported by the local implementation, establish a mapping between the specified payload type and the media format, as described in [RFC3264], Section 6.1. Specifically, this means that the implementation records the payload type to be used in outgoing RTP packets when sending each specified media format, as well as the relative preference for each format that is indicated in their ordering. If any indicated media format is not supported by the local implementation, it MUST be ignored.

* For each specified "rtx" media format, establish a mapping between the RTX payload type and its associated primary payload type, as described in [RFC4588], Section 4. If any referenced primary payload types are not present, this MUST result in an error.

* For each specified fmtp parameter that is supported by the local implementation, enable them on the associated media formats.

* For each specified RTP header extension that is also supported by the local implementation, establish a mapping between the extension ID and URI, as described in [RFC5285], Section 5. Specifically, this means that the implementation records the extension ID to be used in outgoing RTP packets when sending each specified header extension. If any indicated RTP header extension is not supported by the local implementation, it MUST be ignored.

* For each specified RTCP feedback mechanism that is supported by the local implementation, enable them on the associated media formats.

* For any specified "TIAS" bandwidth value, set this value as a constraint on the maximum RTP bitrate to be used when sending media, as specified in [RFC3890]. If a "TIAS" value is not present, but an "AS" value is specified, generate a "TIAS" value using this formula:

  TIAS = AS * 1000 * 0.95 - 50 * 40 * 8

  The 50 is based on 50 packets per second, the 40 is based on an estimate of total header size, the 1000 changes the unit from kbps to bps (as required by TIAS), and the 0.95 is to allocate 5% to RTCP. "TIAS" is used in preference to "AS" because it provides more accurate control of bandwidth.

* For any "RR" or "RS" bandwidth values, handle as specified in [RFC3556], Section 2.

* Any specified "CT" bandwidth value MUST be ignored, as the meaning of this construct at the media level is not well defined.

* If the m= section is of type audio:

  + For each specified "CN" media format, enable DTX for all supported media formats with the same clockrate, as described in [RFC3389], Section 5, except for formats that have their own internal DTX mechanisms. DTX for such formats (e.g., Opus) is controlled via fmtp parameters, as discussed in Section 5.2.3.2.

  + For each specified "telephone-event" media format, enable DTMF transmission for all supported media formats with the same clockrate, as described in [RFC4733], Section 2.5.1.2. If the application attempts to transmit DTMF when using a media format that does not have a corresponding telephone-event format, this MUST result in an error.

  + For any specified "ptime" value, configure the available media formats to use the specified packet size. If the specified size is not supported for a media format, use the next closest value instead.

Finally, if this description is of type "pranswer" or "answer", follow the processing defined in the Section 5.10 section below.

5.10. Applying an Answer

In addition to the steps mentioned above for processing a local or remote description, the following steps are performed when processing a description of type "pranswer" or "answer".

For each m= section, the following steps MUST be performed:

o If the m= section has been rejected (i.e. port is set to zero in the answer), stop any reception or transmission of media for this section, and, unless a non-rejected m= section is bundled with this m= section, discard any associated ICE components, as described in Section 9.2.1.3 of [RFC5245].

o If the remote DTLS fingerprint has been changed or the dtls-id has changed, tear down the DTLS connection. This includes the case when the PeerConnection state is "have-remote-pranswer". If a DTLS connection needs to be torn down but the answer does not indicate an ICE restart or, in the case of "have-remote-pranswer", new ICE credentials, an error MUST be generated. If an ICE restart is performed without a change in dtls-id or fingerprint, then the same DTLS connection is continued over the new ICE channel.

o If no valid DTLS connection exists, prepare to start a DTLS connection, using the specified roles and fingerprints, on any underlying ICE components, once they are active.

o If the m= section proto value indicates use of RTP:

  * If the m= section references any media formats, RTP header extensions, or RTCP feedback mechanisms that were not present in the corresponding m= section in the offer, this indicates a negotiation problem and MUST result in an error.

  * If the m= section has RTCP mux enabled, discard the RTCP ICE component, if one exists, and begin or continue muxing RTCP over the RTP ICE component, as specified in [RFC5761], Section 5.1.3. Otherwise, prepare to transmit RTCP over the RTCP ICE component; if no RTCP ICE component exists, because RTCP mux was previously enabled, this MUST result in an error.

  * If the m= section has reduced-size RTCP enabled, configure the RTCP transmission for this m= section to use reduced-size RTCP, as specified in [RFC5506].

  * If the directional attribute in the answer is of type "sendrecv" or "sendonly", choose the media format to send as

the most preferred media format from the remote description that is also present in the answer, as described in [RFC3264], Sections 6.1 and 7, and start transmitting RTP media once the underlying transport layers have been established. If an SSRC has not already been chosen for this outgoing RTP stream, choose a random one. If media is already being transmitted, the same SSRC SHOULD be used unless the clockrate of the new codec is different, in which case a new SSRC MUST be chosen, as specified in [RFC7160], Section 3.1.

* The payload type mapping from the remote description is used to determine payload types for the outgoing RTP streams, including the payload type for the send media format chosen above. Any RTP header extensions that were negotiated should be included in the outgoing RTP streams, using the extension mapping from the remote description; if the RID header extension has been negotiated, and RID values are specified, include the RID header extension in the outgoing RTP streams, as indicated in [I-D.ietf-mmusic-rid], Section 4.

* If simulcast has been negotiated, send the number of Source RTP Streams as specified in [I-D.ietf-mmusic-sdp-simulcast], Section 6.2.2.

* If the send media format chosen above has a corresponding "rtx" media format, or a FEC mechanism has been negotiated, establish a Redundancy RTP Stream with a random SSRC for each Source RTP Stream, and start or continue transmitting RTX/FEC packets as needed.

* If the send media format chosen above has a corresponding "red" media format of the same clockrate, allow redundant encoding using the specified format for resiliency purposes, as discussed in [I-D.ietf-rtcweb-fec], Section 3.2. Note that unlike RTX or FEC media formats, the "red" format is transmitted on the Source RTP Stream, not the Redundancy RTP Stream.

* Enable the RTCP feedback mechanisms referenced in the media section for all Source RTP Streams using the specified media formats. Specifically, begin or continue sending the requested feedback types and reacting to received feedback, as specified in [RFC4585], Section 4.2. When sending RTCP feedback, follow the rules and recommendations from [I-D.ietf-avtcore-rtp-multi-stream], Section 5.4.1 to select which SSRC to use.

   * If the directional attribute is of type "recvonly" or "inactive", stop transmitting all RTP media, but continue sending RTCP, as described in [RFC3264], Section 5.1.

o If the m= section proto value indicates use of SCTP:

   * If an SCTP association exists, and the remote SCTP port has changed, discard the existing SCTP association. This includes the case when the PeerConnection state is "have-remote-pranswer".

   * If no valid SCTP association exists, prepare to initiate a SCTP association over the associated ICE component and DTLS connection, using the local SCTP port value from the local description, and the remote SCTP port value from the remote description, as described in [I-D.ietf-mmusic-sctp-sdp], Section 10.2.

If the answer contains valid bundle groups, discard any ICE components for the m= sections that will be bundled onto the primary ICE components in each bundle, and begin muxing these m= sections accordingly, as described in [I-D.ietf-mmusic-sdp-bundle-negotiation], Section 8.2.

If the description is of type "answer", and there are still remaining candidates in the ICE candidate pool, discard them.

6. Processing RTP/RTCP

When bundling, associating incoming RTP/RTCP with the proper m= section is defined in [I-D.ietf-mmusic-sdp-bundle-negotiation]. [The BUNDLE draft does not currently contain the necessary text to describe this demux, but when it does it will contain text like that contained in Appendix B.] When not bundling, the proper m= section is clear from the ICE component over which the RTP/RTCP is received.

Once the proper m= section(s) are known, RTP/RTCP is delivered to the RtpTransceiver(s) associated with the m= section(s) and further processing of the RTP/RTCP is done at the RtpTransceiver level. This includes using RID [I-D.ietf-mmusic-rid] to distinguish between multiple Encoded Streams, as well as determine which Source RTP stream should be repaired by a given Redundancy RTP stream.

7. Examples

Note that this example section shows several SDP fragments. To format in 72 columns, some of the lines in SDP have been split into multiple lines, where leading whitespace indicates that a line is a

continuation of the previous line. In addition, some blank lines have been added to improve readability but are not valid in SDP.

More examples of SDP for WebRTC call flows can be found in [I-D.nandakumar-rtcweb-sdp].

7.1. Simple Example

This section shows a very simple example that sets up a minimal audio / video call between two JSEP endpoints without using trickle ICE. The example in the following section provides a more detailed example of what could happen in a JSEP session.

The code flow below shows Alice's endpoint initiating the session to Bob's endpoint. The messages from Alice's JS to Bob's JS are assumed to flow over some signaling protocol via a web server. The JS on both Alice's side and Bob's side waits for all candidates before sending the offer or answer, so the offers and answers are complete; trickle ICE is not used. Both Alice and Bob are using the default bundle policy of "balanced", and the default RTCP mux policy of "require".

```
//                    set up local media state
AliceJS->AliceUA:     create new PeerConnection
AliceJS->AliceUA:     addTrack with two tracks: audio and video
AliceJS->AliceUA:     createOffer to get offer
AliceJS->AliceUA:     setLocalDescription with offer
AliceUA->AliceJS:     multiple onicecandidate events with candidates

//                    wait for ICE gathering to complete
AliceUA->AliceJS:     onicecandidate event with null candidate
AliceJS->AliceUA:     get |offer-A1| from pendingLocalDescription

//                    |offer-A1| is sent over signaling protocol to Bob
AliceJS->WebServer:   signaling with |offer-A1|
WebServer->BobJS:     signaling with |offer-A1|

//                    |offer-A1| arrives at Bob
BobJS->BobUA:         create a PeerConnection
BobJS->BobUA:         setRemoteDescription with |offer-A1|
BobUA->BobJS:         ontrack events for audio and video tracks

//                    Bob accepts call
BobJS->BobUA:         addTrack with local tracks
BobJS->BobUA:         createAnswer
BobJS->BobUA:         setLocalDescription with answer
BobUA->BobJS:         multiple onicecandidate events with candidates

//                    wait for ICE gathering to complete
BobUA->BobJS:         onicecandidate event with null candidate
BobJS->BobUA:         get |answer-A1| from currentLocalDescription

//                    |answer-A1| is sent over signaling protocol to Alice
BobJS->WebServer:     signaling with |answer-A1|
WebServer->AliceJS:   signaling with |answer-A1|

//                    |answer-A1| arrives at Alice
AliceJS->AliceUA:     setRemoteDescription with |answer-A1|
AliceUA->AliceJS:     ontrack events for audio and video tracks

//                    media flows
BobUA->AliceUA:       media sent from Bob to Alice
AliceUA->BobUA:       media sent from Alice to Bob
```

The SDP for |offer-A1| looks like:

```
v=0
o=- 4962303333179871722 1 IN IP4 0.0.0.0
```

```
s=-
t=0 0
a=ice-options:trickle
a=group:BUNDLE a1 v1
a=group:LS a1 v1

m=audio 10100 UDP/TLS/RTP/SAVPF 96 0 8 97 98
c=IN IP4 203.0.113.100
a=mid:a1
a=sendrecv
a=rtpmap:96 opus/48000/2
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:97 telephone-event/8000
a=rtpmap:98 telephone-event/48000
a=maxptime:120
a=extmap:1 urn:ietf:params:rtp-hdrext:sdes:mid
a=extmap:2 urn:ietf:params:rtp-hdrext:ssrc-audio-level
a=msid:47017fee-b6c1-4162-929c-a25110252400
       f83006c5-a0ff-4e0a-9ed9-d3e6747be7d9
a=ice-ufrag:ETEn
a=ice-pwd:OtSK0WpNtpUjkY4+86js7ZQl
a=fingerprint:sha-256
              19:E2:1C:3B:4B:9F:81:E6:B8:5C:F4:A5:A8:D8:73:04:
              BB:05:2F:70:9F:04:A9:0E:05:E9:26:33:E8:70:88:A2
a=setup:actpass
a=dtls-id:1
a=rtcp:10101 IN IP4 203.0.113.100
a=rtcp-mux
a=rtcp-rsize
a=candidate:1 1 udp 2113929471 203.0.113.100 10100 typ host
a=candidate:1 2 udp 2113929470 203.0.113.100 10101 typ host
a=end-of-candidates

m=video 10102 UDP/TLS/RTP/SAVPF 100 101
c=IN IP4 203.0.113.100
a=mid:v1
a=sendrecv
a=rtpmap:100 VP8/90000
a=rtpmap:101 rtx/90000
a=fmtp:101 apt=100
a=extmap:1 urn:ietf:params:rtp-hdrext:sdes:mid
a=rtcp-fb:100 ccm fir
a=rtcp-fb:100 nack
a=rtcp-fb:100 nack pli
a=msid:47017fee-b6c1-4162-929c-a25110252400
       f30bdb4a-5db8-49b5-bcdc-e0c9a23172e0
a=ice-ufrag:BGKk
```

```
a=ice-pwd:mqyWsAjvtKwTGnvhPztQ9mIf
a=fingerprint:sha-256
              19:E2:1C:3B:4B:9F:81:E6:B8:5C:F4:A5:A8:D8:73:04:
              BB:05:2F:70:9F:04:A9:0E:05:E9:26:33:E8:70:88:A2
a=setup:actpass
a=dtls-id:1
a=rtcp:10103 IN IP4 203.0.113.100
a=rtcp-mux
a=rtcp-rsize
a=candidate:1 1 udp 2113929471 203.0.113.100 10102 typ host
a=candidate:1 2 udp 2113929470 203.0.113.100 10103 typ host
a=end-of-candidates
```

The SDP for |answer-A1| looks like:

```
v=0
o=- 6729291447651054566 1 IN IP4 0.0.0.0
s=-
t=0 0
a=ice-options:trickle
a=group:BUNDLE a1 v1
a=group:LS a1 v1

m=audio 10200 UDP/TLS/RTP/SAVPF 96 0 8 97 98
c=IN IP4 203.0.113.200
a=mid:a1
a=sendrecv
a=rtpmap:96 opus/48000/2
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:97 telephone-event/8000
a=rtpmap:98 telephone-event/48000
a=maxptime:120
a=extmap:1 urn:ietf:params:rtp-hdrext:sdes:mid
a=extmap:2 urn:ietf:params:rtp-hdrext:ssrc-audio-level
a=msid:61317484-2ed4-49d7-9eb7-1414322a7aae
       5a7b57b8-f043-4bd1-a45d-09d4dfa31226
a=ice-ufrag:6sFv
a=ice-pwd:cOTZKZNVlO9RSGsEGM63JXT2
a=fingerprint:sha-256
              6B:8B:F0:65:5F:78:E2:51:3B:AC:6F:F3:3F:46:1B:35:
              DC:B8:5F:64:1A:24:C2:43:F0:A1:58:D0:A1:2C:19:08
a=setup:active
a=dtls-id:1
a=rtcp-mux
a=rtcp-rsize
a=candidate:1 1 udp 2113929471 203.0.113.200 10200 typ host
a=end-of-candidates

m=video 10200 UDP/TLS/RTP/SAVPF 100 101
c=IN IP4 203.0.113.200
a=mid:v1
a=sendrecv
a=rtpmap:100 VP8/90000
a=rtpmap:101 rtx/90000
a=fmtp:101 apt=100
a=extmap:1 urn:ietf:params:rtp-hdrext:sdes:mid
a=rtcp-fb:100 ccm fir
a=rtcp-fb:100 nack
a=rtcp-fb:100 nack pli
a=msid:61317484-2ed4-49d7-9eb7-1414322a7aae
       4ea4d4a1-2fda-4511-a9cc-1b32c2e59552
```

7.2. Detailed Example

This section shows a more involved example of a session between two JSEP endpoints. Trickle ICE is used in full trickle mode, with a bundle policy of "max-bundle", an RTCP mux policy of "require", and a single TURN server. Initially, both Alice and Bob establish an audio channel and a data channel. Later, Bob adds two video flows, one for his video feed, and one for screensharing, both supporting FEC, and with the video feed configured for simulcast. Alice accepts these video flows, but does not add video flows of her own, so they are handled as recvonly. Alice also specifies a maximum video decoder resolution.

```
//                    set up local media state
AliceJS->AliceUA:   create new PeerConnection
AliceJS->AliceUA:   addTrack with an audio track
AliceJS->AliceUA:   createDataChannel to get data channel
AliceJS->AliceUA:   createOffer to get |offer-B1|
AliceJS->AliceUA:   setLocalDescription with |offer-B1|

//                    |offer-B1| is sent over signaling protocol to Bob
AliceJS->WebServer: signaling with |offer-B1|
WebServer->BobJS:   signaling with |offer-B1|

//                    |offer-B1| arrives at Bob
BobJS->BobUA:       create a PeerConnection
BobJS->BobUA:       setRemoteDescription with |offer-B1|
BobUA->BobJS:       ontrack with audio track from Alice

//                    candidates are sent to Bob
AliceUA->AliceJS:   onicecandidate (host) |offer-B1-candidate-1|
AliceJS->WebServer: signaling with |offer-B1-candidate-1|
AliceUA->AliceJS:   onicecandidate (srflx) |offer-B1-candidate-2|
AliceJS->WebServer: signaling with |offer-B1-candidate-2|
AliceUA->AliceJS:   onicecandidate (relay) |offer-B1-candidate-3|
AliceJS->WebServer: signaling with |offer-B1-candidate-3|

WebServer->BobJS:   signaling with |offer-B1-candidate-1|
BobJS->BobUA:       addIceCandidate with |offer-B1-candidate-1|
WebServer->BobJS:   signaling with |offer-B1-candidate-2|
BobJS->BobUA:       addIceCandidate with |offer-B1-candidate-2|
WebServer->BobJS:   signaling with |offer-B1-candidate-3|
BobJS->BobUA:       addIceCandidate with |offer-B1-candidate-3|

//                    Bob accepts call
BobJS->BobUA:       addTrack with local audio
BobJS->BobUA:       createDataChannel to get data channel
```

```
BobJS->BobUA:        createAnswer to get |answer-B1|
BobJS->BobUA:        setLocalDescription with |answer-B1|

//                   |answer-B1| is sent to Alice
BobJS->WebServer:    signaling with |answer-B1|
WebServer->AliceJS: signaling with |answer-B1|
AliceJS->AliceUA:    setRemoteDescription with |answer-B1|
AliceUA->AliceJS:    ontrack event with audio track from Bob

//                   candidates are sent to Alice
BobUA->BobJS:        onicecandidate (host) |answer-B1-candidate-1|
BobJS->WebServer:    signaling with |answer-B1-candidate-1|
BobUA->BobJS:        onicecandidate (srflx) |answer-B1-candidate-2|
BobJS->WebServer:    signaling with |answer-B1-candidate-2|
BobUA->BobJS:        onicecandidate (relay) |answer-B1-candidate-3|
BobJS->WebServer:    signaling with |answer-B1-candidate-3|

WebServer->AliceJS: signaling with |answer-B1-candidate-1|
AliceJS->AliceUA:    addIceCandidate with |answer-B1-candidate-1|
WebServer->AliceJS: signaling with |answer-B1-candidate-2|
AliceJS->AliceUA:    addIceCandidate with |answer-B1-candidate-2|
WebServer->AliceJS: signaling with |answer-B1-candidate-3|
AliceJS->AliceUA:    addIceCandidate with |answer-B1-candidate-3|

//                   data channel opens
BobUA->BobJS:        ondatachannel event
AliceUA->AliceJS:    ondatachannel event
BobUA->BobJS:        onopen
AliceUA->AliceJS:    onopen

//                   media is flowing between endpoints
BobUA->AliceUA:      audio+data sent from Bob to Alice
AliceUA->BobUA:      audio+data sent from Alice to Bob

//                   some time later Bob adds two video streams
//                   note, no candidates exchanged, because of bundle
BobJS->BobUA:        addTrack with first video stream
BobJS->BobUA:        addTrack with second video stream
BobJS->BobUA:        createOffer to get |offer-B2|
BobJS->BobUA:        setLocalDescription with |offer-B2|

//                   |offer-B2| is sent to Alice
BobJS->WebServer:    signaling with |offer-B2|
WebServer->AliceJS: signaling with |offer-B2|
AliceJS->AliceUA:    setRemoteDescription with |offer-B2|
AliceUA->AliceJS:    ontrack event with first video track
AliceUA->AliceJS:    ontrack event with second video track
AliceJS->AliceUA:    createAnswer to get |answer-B2|
```

```
AliceJS->AliceUA:   setLocalDescription with |answer-B2|

//                  |answer-B2| is sent over signaling protocol to Bob
AliceJS->WebServer: signaling with |answer-B2|
WebServer->BobJS:   signaling with |answer-B2|
BobJS->BobUA:       setRemoteDescription with |answer-B2|

//                  media is flowing between endpoints
BobUA->AliceUA:     audio+video+data sent from Bob to Alice
AliceUA->BobUA:     audio+video+data sent from Alice to Bob
```

The SDP for |offer-B1| looks like:

```
v=0
o=- 4962303333179871723 1 IN IP4 0.0.0.0
s=-
t=0 0
a=ice-options:trickle
a=group:BUNDLE a1 d1

m=audio 9 UDP/TLS/RTP/SAVPF 96 0 8 97 98
c=IN IP4 0.0.0.0
a=mid:a1
a=sendrecv
a=rtpmap:96 opus/48000/2
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:97 telephone-event/8000
a=rtpmap:98 telephone-event/48000
a=maxptime:120
a=extmap:1 urn:ietf:params:rtp-hdrext:sdes:mid
a=extmap:2 urn:ietf:params:rtp-hdrext:ssrc-audio-level
a=msid:57017fee-b6c1-4162-929c-a25110252400
       e83006c5-a0ff-4e0a-9ed9-d3e6747be7d9
a=ice-ufrag:ATEn
a=ice-pwd:AtSK0WpNtpUjkY4+86js7ZQl
a=fingerprint:sha-256
              29:E2:1C:3B:4B:9F:81:E6:B8:5C:F4:A5:A8:D8:73:04:
              BB:05:2F:70:9F:04:A9:0E:05:E9:26:33:E8:70:88:A2
a=setup:actpass
a=dtls-id:1
a=rtcp-mux
a=rtcp-mux-only
a=rtcp-rsize

m=application 0 UDP/DTLS/SCTP webrtc-datachannel
c=IN IP4 0.0.0.0
a=mid:d1
a=sctp-port:5000
a=max-message-size:65536
a=bundle-only
```

|offer-B1-candidate-1| looks like:

```
ufrag ATEn
index 0
mid   a1
attr  candidate:1 1 udp 2113929471 203.0.113.100 10100 typ host
```

|offer-B1-candidate-2| looks like:

```
ufrag ATEn
index 0
mid   a1
attr  candidate:1 1 udp 1845494015 198.51.100.100 11100 typ srflx
                raddr 203.0.113.100 rport 10100
```

|offer-B1-candidate-3| looks like:

```
ufrag ATEn
index 0
mid   a1
attr  candidate:1 1 udp 255 192.0.2.100 12100 typ relay
                raddr 198.51.100.100 rport 11100
```

The SDP for |answer-B1| looks like:

```
v=0
o=- 7729291447651054566 1 IN IP4 0.0.0.0
s=-
t=0 0
a=ice-options:trickle
a=group:BUNDLE a1 d1

m=audio 9 UDP/TLS/RTP/SAVPF 96 0 8 97 98
c=IN IP4 0.0.0.0
a=mid:a1
a=sendrecv
a=rtpmap:96 opus/48000/2
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:97 telephone-event/8000
a=rtpmap:98 telephone-event/48000
a=maxptime:120
a=extmap:1 urn:ietf:params:rtp-hdrext:sdes:mid
a=extmap:2 urn:ietf:params:rtp-hdrext:ssrc-audio-level
a=msid:71317484-2ed4-49d7-9eb7-1414322a7aae
       6a7b57b8-f043-4bd1-a45d-09d4dfa31226
a=ice-ufrag:7sFv
a=ice-pwd:dOTZKZNVlO9RSGsEGM63JXT2
a=fingerprint:sha-256
              7B:8B:F0:65:5F:78:E2:51:3B:AC:6F:F3:3F:46:1B:35:
              DC:B8:5F:64:1A:24:C2:43:F0:A1:58:D0:A1:2C:19:08
a=setup:active
a=dtls-id:1
a=rtcp-mux
a=rtcp-mux-only
a=rtcp-rsize

m=application 9 UDP/DTLS/SCTP webrtc-datachannel
c=IN IP4 0.0.0.0
a=mid:d1
a=sctp-port:5000
a=max-message-size:65536
```

|answer-B1-candidate-1| looks like:

```
ufrag 7sFv
index 0
mid   a1
attr  candidate:1 1 udp 2113929471 203.0.113.200 10200 typ host
```

|answer-B1-candidate-2| looks like:

```
ufrag 7sFv
index 0
mid   a1
attr  candidate:1 1 udp 1845494015 198.51.100.200 11200 typ srflx
                raddr 203.0.113.200 rport 10200
```

|answer-B1-candidate-3| looks like:

```
ufrag 7sFv
index 0
mid   a1
attr  candidate:1 1 udp 255 192.0.2.200 12200 typ relay
                raddr 198.51.100.200 rport 11200
```

The SDP for |offer-B2| is shown below. In addition to the new m= sections for video, both of which are offering FEC, and one of which is offering simulcast, note the increment of the version number in the o= line, changes to the c= line, indicating the local candidate that was selected, and the inclusion of gathered candidates as a=candidate lines.

```
v=0
o=- 7729291447651054566 2 IN IP4 0.0.0.0
s=-
t=0 0
a=ice-options:trickle
a=group:BUNDLE a1 d1 v1 v2
a=group:LS a1 v1

m=audio 12200 UDP/TLS/RTP/SAVPF 96 0 8 97 98
c=IN IP4 192.0.2.200
a=mid:a1
a=sendrecv
a=rtpmap:96 opus/48000/2
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:97 telephone-event/8000
a=rtpmap:98 telephone-event/48000
a=maxptime:120
a=extmap:1 urn:ietf:params:rtp-hdrext:sdes:mid
a=extmap:2 urn:ietf:params:rtp-hdrext:ssrc-audio-level
```

```
   a=msid:71317484-2ed4-49d7-9eb7-1414322a7aae
          6a7b57b8-f043-4bd1-a45d-09d4dfa31226
   a=ice-ufrag:7sFv
   a=ice-pwd:dOTZKZNVlO9RSGsEGM63JXT2
   a=fingerprint:sha-256
                 7B:8B:F0:65:5F:78:E2:51:3B:AC:6F:F3:3F:46:1B:35:
                 DC:B8:5F:64:1A:24:C2:43:F0:A1:58:D0:A1:2C:19:08
   a=setup:actpass
   a=dtls-id:1
   a=rtcp-mux
   a=rtcp-mux-only
   a=rtcp-rsize
   a=candidate:1 1 udp 2113929471 203.0.113.200 10200 typ host
   a=candidate:1 1 udp 1845494015 198.51.100.200 11200 typ srflx
               raddr 203.0.113.200 rport 10200
   a=candidate:1 1 udp 255 192.0.2.200 12200 typ relay
               raddr 198.51.100.200 rport 11200
   a=end-of-candidates

   m=application 12200 UDP/DTLS/SCTP webrtc-datachannel
   c=IN IP4 192.0.2.200
   a=mid:d1
   a=sctp-port:5000
   a=max-message-size:65536

   m=video 12200 UDP/TLS/RTP/SAVPF 100 101 102
   c=IN IP4 192.0.2.200
   a=mid:v1
   a=sendrecv
   a=rtpmap:100 VP8/90000
   a=rtpmap:101 rtx/90000
   a=fmtp:101 apt=100
   a=rtpmap:102 flexfec/90000
   a=extmap:1 urn:ietf:params:rtp-hdrext:sdes:mid
   a=rtcp-fb:100 ccm fir
   a=rtcp-fb:100 nack
   a=rtcp-fb:100 nack pli
   a=msid:71317484-2ed4-49d7-9eb7-1414322a7aae
          5ea4d4a1-2fda-4511-a9cc-1b32c2e59552
   a=rid:1 send
   a=rid:2 send
   a=rid:3 send
   a=simulcast:send 1;2;3

   m=video 12200 UDP/TLS/RTP/SAVPF 100 101 102
   c=IN IP4 192.0.2.200
   a=mid:v2
   a=sendrecv
```

```
a=rtpmap:100 VP8/90000
a=rtpmap:101 rtx/90000
a=fmtp:101 apt=100
a=rtpmap:102 flexfec/90000
a=extmap:1 urn:ietf:params:rtp-hdrext:sdes:mid
a=rtcp-fb:100 ccm fir
a=rtcp-fb:100 nack
a=rtcp-fb:100 nack pli
a=msid:81317484-2ed4-49d7-9eb7-1414322a7aae
       6ea4d4a1-2fda-4511-a9cc-1b32c2e59552
```

The SDP for |answer-B2| is shown below. In addition to the acceptance of the video m= sections, the use of a=recvonly to indicate one-way video, and the use of a=imageattr to limit the received resolution, note the use of setup:passive to maintain the existing DTLS roles.

```
v=0
o=- 4962303333179871723 2 IN IP4 0.0.0.0
s=-
t=0 0
a=ice-options:trickle
a=group:BUNDLE a1 d1 v1 v2
a=group:LS a1 v1

m=audio 12100 UDP/TLS/RTP/SAVPF 96 0 8 97 98
c=IN IP4 192.0.2.100
a=mid:a1
a=sendrecv
a=rtpmap:96 opus/48000/2
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:97 telephone-event/8000
a=rtpmap:98 telephone-event/48000
a=maxptime:120
a=extmap:1 urn:ietf:params:rtp-hdrext:sdes:mid
a=extmap:2 urn:ietf:params:rtp-hdrext:ssrc-audio-level
a=msid:57017fee-b6c1-4162-929c-a25110252400
       e83006c5-a0ff-4e0a-9ed9-d3e6747be7d9
a=ice-ufrag:ATEn
a=ice-pwd:AtSK0WpNtpUjkY4+86js7ZQl
a=fingerprint:sha-256
              29:E2:1C:3B:4B:9F:81:E6:B8:5C:F4:A5:A8:D8:73:04:
              BB:05:2F:70:9F:04:A9:0E:05:E9:26:33:E8:70:88:A2
a=setup:passive
a=dtls-id:1
```

```
a=rtcp-mux
a=rtcp-mux-only
a=rtcp-rsize
a=candidate:1 1 udp 2113929471 203.0.113.100 10100 typ host
a=candidate:1 1 udp 1845494015 198.51.100.100 11100 typ srflx
            raddr 203.0.113.100 rport 10100
a=candidate:1 1 udp 255 192.0.2.100 12100 typ relay
            raddr 198.51.100.100 rport 11100
a=end-of-candidates

m=application 12100 UDP/DTLS/SCTP webrtc-datachannel
c=IN IP4 192.0.2.100
a=mid:d1
a=sctp-port:5000
a=max-message-size:65536

m=video 12100 UDP/TLS/RTP/SAVPF 100 101
c=IN IP4 192.0.2.100
a=mid:v1
a=recvonly
a=rtpmap:100 VP8/90000
a=rtpmap:101 rtx/90000
a=fmtp:101 apt=100
a=imageattr:100 recv [x=[48:1920],y=[48:1080],q=1.0]
a=extmap:1 urn:ietf:params:rtp-hdrext:sdes:mid
a=rtcp-fb:100 ccm fir
a=rtcp-fb:100 nack
a=rtcp-fb:100 nack pli

m=video 12100 UDP/TLS/RTP/SAVPF 100 101
c=IN IP4 192.0.2.100
a=mid:v2
a=recvonly
a=rtpmap:100 VP8/90000
a=rtpmap:101 rtx/90000
a=fmtp:101 apt=100
a=imageattr:100 recv [x=[48:1920],y=[48:1080],q=1.0]
a=extmap:1 urn:ietf:params:rtp-hdrext:sdes:mid
a=rtcp-fb:100 ccm fir
a=rtcp-fb:100 nack
a=rtcp-fb:100 nack pli
```

7.3. Early Transport Warmup Example

This example demonstrates the early warmup technique described in Section 4.1.8.1. Here, Alice's endpoint sends an offer to Bob's endpoint to start an audio/video call. Bob immediately responds with

an answer that accepts the audio/video m= sections, but marks them as sendonly (from his perspective), meaning that Alice will not yet send media. This allows the JSEP implementation to start negotiating ICE and DTLS immediately. Bob's endpoint then prompts him to answer the call, and when he does, his endpoint sends a second offer which enables the audio and video m= sections, and thereby bidirectional media transmission. The advantage of such a flow is that as soon as the first answer is received, the implementation can proceed with ICE and DTLS negotiation and establish the session transport. If the transport setup completes before the second offer is sent, then media can be transmitted immediately by the callee immediately upon answering the call, minimizing perceived post-dial-delay. The second offer/answer exchange can also change the preferred codecs or other session parameters.

This example also makes use of the "relay" ICE candidate policy described in Section 3.5.3 to minimize the ICE gathering and checking needed.

```
//                    set up local media state
AliceJS->AliceUA:   create new PeerConnection with "relay" ICE policy
AliceJS->AliceUA:   addTrack with two tracks: audio and video
AliceJS->AliceUA:   createOffer to get |offer-C1|
AliceJS->AliceUA:   setLocalDescription with |offer-C1|

//                    |offer-C1| is sent over signaling protocol to Bob
AliceJS->WebServer: signaling with |offer-C1|
WebServer->BobJS:   signaling with |offer-C1|

//                    |offer-C1| arrives at Bob
BobJS->BobUA:       create new PeerConnection with "relay" ICE policy
BobJS->BobUA:       setRemoteDescription with |offer-C1|
BobUA->BobJS:       ontrack events for audio and video

//                    a relay candidate is sent to Bob
AliceUA->AliceJS:   onicecandidate (relay) |offer-C1-candidate-1|
AliceJS->WebServer: signaling with |offer-C1-candidate-1|

WebServer->BobJS:   signaling with |offer-C1-candidate-1|
BobJS->BobUA:       addIceCandidate with |offer-C1-candidate-1|

//                    Bob prepares an early answer to warm up the transport
BobJS->BobUA:       addTransceiver with null audio and video tracks
BobJS->BobUA:       transceiver.setDirection(sendonly) for both
BobJS->BobUA:       createAnswer
BobJS->BobUA:       setLocalDescription with answer

//                    |answer-C1| is sent over signaling protocol to Alice
```

```
BobJS->WebServer:   signaling with |answer-C1|
WebServer->AliceJS: signaling with |answer-C1|

//                  |answer-C1| (sendonly) arrives at Alice
AliceJS->AliceUA:   setRemoteDescription with |answer-C1|
AliceUA->AliceJS:   ontrack events for audio and video

//                  a relay candidate is sent to Alice
BobUA->BobJS:       onicecandidate (relay) |answer-B1-candidate-1|
BobJS->WebServer:   signaling with |answer-B1-candidate-1|

WebServer->AliceJS: signaling with |answer-B1-candidate-1|
AliceJS->AliceUA:   addIceCandidate with |answer-B1-candidate-1|

//                  ICE and DTLS establish while call is ringing

//                 Bob accepts call, starts media, and sends a new offer
BobJS->BobUA:       transceiver.setTrack with audio and video tracks
BobUA->AliceUA:     media sent from Bob to Alice
BobJS->BobUA:       transceiver.setDirection(sendrecv) for both
                    transceivers
BobJS->BobUA:       createOffer
BobJS->BobUA:       setLocalDescription with offer

//                  |offer-C2| is sent over signaling protocol to Alice
BobJS->WebServer:   signaling with |offer-C2|
WebServer->AliceJS: signaling with |offer-C2|

//                  |offer-C2| (sendrecv) arrives at Alice
AliceJS->AliceUA:   setRemoteDescription with |offer-C2|
AliceJS->AliceUA:   createAnswer
AliceJS->AliceUA:   setLocalDescription with |answer-C2|
AliceUA->BobUA:     media sent from Alice to Bob

//                  |answer-C2| is sent over signaling protocol to Bob
AliceJS->WebServer: signaling with |answer-C2|
WebServer->BobJS:   signaling with |answer-C2|
BobJS->BobUA:       setRemoteDescription with |answer-C2|
```

The SDP for |offer-C1| looks like:

```
v=0
o=- 1070771854436052752 1 IN IP4 0.0.0.0
s=-
t=0 0
a=ice-options:trickle
a=group:BUNDLE a1 v1
a=group:LS a1 v1

m=audio 9 UDP/TLS/RTP/SAVPF 96 0 8 97 98
c=IN IP4 0.0.0.0
a=mid:a1
a=sendrecv
a=rtpmap:96 opus/48000/2
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:97 telephone-event/8000
a=rtpmap:98 telephone-event/48000
a=maxptime:120
a=extmap:1 urn:ietf:params:rtp-hdrext:sdes:mid
a=extmap:2 urn:ietf:params:rtp-hdrext:ssrc-audio-level
a=msid:bbce3ba6-abfc-ac63-d00a-e15b286f8fce
       e80098db-7159-3c06-229a-00df2a9b3dbc
a=ice-ufrag:4ZcD
a=ice-pwd:ZaaG6OG7tCn4J/lehAGz+HHD
a=fingerprint:sha-256
              C4:68:F8:77:6A:44:F1:98:6D:7C:9F:47:EB:E3:34:A4:
              0A:AA:2D:49:08:28:70:2E:1F:AE:18:7D:4E:3E:66:BF
a=setup:actpass
a=dtls-id:1
a=rtcp-mux
a=rtcp-mux-only
a=rtcp-rsize

m=video 0 UDP/TLS/RTP/SAVPF 100 101
c=IN IP4 0.0.0.0
a=mid:v1
a=sendrecv
a=rtpmap:100 VP8/90000
a=rtpmap:101 rtx/90000
a=fmtp:101 apt=100
a=extmap:1 urn:ietf:params:rtp-hdrext:sdes:mid
a=rtcp-fb:100 ccm fir
a=rtcp-fb:100 nack
a=rtcp-fb:100 nack pli
a=msid:bbce3ba6-abfc-ac63-d00a-e15b286f8fce
       ac701365-eb06-42df-cc93-7f22bc308789
a=bundle-only
```

|offer-C1-candidate-1| looks like:

```
ufrag 4ZcD
index 0
mid   a1
attr  candidate:1 1 udp 255 192.0.2.100 12100 typ relay
                raddr 0.0.0.0 rport 0
```

The SDP for |answer-C1| looks like:

```
v=0
o=- 6386516489780559513 1 IN IP4 0.0.0.0
s=-
t=0 0
a=ice-options:trickle
a=group:BUNDLE a1 v1
a=group:LS a1 v1

m=audio 9 UDP/TLS/RTP/SAVPF 96 0 8 97 98
c=IN IP4 0.0.0.0
a=mid:a1
a=sendonly
a=rtpmap:96 opus/48000/2
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:97 telephone-event/8000
a=rtpmap:98 telephone-event/48000
a=maxptime:120
a=extmap:1 urn:ietf:params:rtp-hdrext:sdes:mid
a=extmap:2 urn:ietf:params:rtp-hdrext:ssrc-audio-level
a=msid:751f239e-4ae0-c549-aa3d-890de772998b
       04b5a445-82cc-c9e8-9ffe-c24d0ef4b0ff
a=ice-ufrag:TpaA
a=ice-pwd:t2Ouhc67y8JcCaYZxUUTgKw/
a=fingerprint:sha-256
              A2:F3:A5:6D:4C:8C:1E:B2:62:10:4A:F6:70:61:C4:FC:
              3C:E0:01:D6:F3:24:80:74:DA:7C:3E:50:18:7B:CE:4D
a=setup:active
a=dtls-id:1
a=rtcp-mux
a=rtcp-mux-only
a=rtcp-rsize

m=video 9 UDP/TLS/RTP/SAVPF 100 101
c=IN IP4 0.0.0.0
a=mid:v1
a=sendonly
a=rtpmap:100 VP8/90000
a=rtpmap:101 rtx/90000
a=fmtp:101 apt=100
a=extmap:1 urn:ietf:params:rtp-hdrext:sdes:mid
a=rtcp-fb:100 ccm fir
a=rtcp-fb:100 nack
a=rtcp-fb:100 nack pli
a=msid:751f239e-4ae0-c549-aa3d-890de772998b
       39292672-c102-d075-f580-5826f31ca958
```

|answer-C1-candidate-1| looks like:

```
ufrag TpaA
index 0
mid   a1
attr  candidate:1 1 udp 255 192.0.2.200 12200 typ relay
                raddr 0.0.0.0 rport 0
```

The SDP for |offer-C2| looks like:

```
v=0
o=- 6386516489780559513 2 IN IP4 0.0.0.0
s=-
t=0 0
a=ice-options:trickle
a=group:BUNDLE a1 v1
a=group:LS a1 v1

m=audio 12200 UDP/TLS/RTP/SAVPF 96 0 8 97 98
c=IN IP4 192.0.2.200
a=mid:a1
a=sendrecv
a=rtpmap:96 opus/48000/2
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:97 telephone-event/8000
a=rtpmap:98 telephone-event/48000
a=maxptime:120
a=extmap:1 urn:ietf:params:rtp-hdrext:sdes:mid
a=extmap:2 urn:ietf:params:rtp-hdrext:ssrc-audio-level
a=msid:751f239e-4ae0-c549-aa3d-890de772998b
       04b5a445-82cc-c9e8-9ffe-c24d0ef4b0ff
a=ice-ufrag:TpaA
a=ice-pwd:t2Ouhc67y8JcCaYZxUUTgKw/
a=fingerprint:sha-256
              A2:F3:A5:6D:4C:8C:1E:B2:62:10:4A:F6:70:61:C4:FC:
              3C:E0:01:D6:F3:24:80:74:DA:7C:3E:50:18:7B:CE:4D
a=setup:actpass
a=dtls-id:1
a=rtcp-mux
a=rtcp-mux-only
a=rtcp-rsize
a=candidate:1 1 udp 255 192.0.2.200 12200 typ relay
            raddr 0.0.0.0 rport 0
a=end-of-candidates
```

```
m=video 12200 UDP/TLS/RTP/SAVPF 100 101
c=IN IP4 192.0.2.200
a=mid:v1
a=sendrecv
a=rtpmap:100 VP8/90000
a=rtpmap:101 rtx/90000
a=fmtp:101 apt=100
a=extmap:1 urn:ietf:params:rtp-hdrext:sdes:mid
a=rtcp-fb:100 ccm fir
a=rtcp-fb:100 nack
a=rtcp-fb:100 nack pli
a=msid:751f239e-4ae0-c549-aa3d-890de772998b
       39292672-c102-d075-f580-5826f31ca958
```

The SDP for |answer-C2| looks like:

```
v=0
o=- 1070771854436052752 2 IN IP4 0.0.0.0
s=-
t=0 0
a=ice-options:trickle
a=group:BUNDLE a1 v1
a=group:LS a1 v1

m=audio 12100 UDP/TLS/RTP/SAVPF 96 0 8 97 98
c=IN IP4 192.0.2.100
a=mid:a1
a=sendrecv
a=rtpmap:96 opus/48000/2
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:97 telephone-event/8000
a=rtpmap:98 telephone-event/48000
a=maxptime:120
a=extmap:1 urn:ietf:params:rtp-hdrext:sdes:mid
a=extmap:2 urn:ietf:params:rtp-hdrext:ssrc-audio-level
a=msid:bbce3ba6-abfc-ac63-d00a-e15b286f8fce
       e80098db-7159-3c06-229a-00df2a9b3dbc
a=ice-ufrag:4ZcD
a=ice-pwd:ZaaG6OG7tCn4J/lehAGz+HHD
a=fingerprint:sha-256
              C4:68:F8:77:6A:44:F1:98:6D:7C:9F:47:EB:E3:34:A4:
              0A:AA:2D:49:08:28:70:2E:1F:AE:18:7D:4E:3E:66:BF
a=setup:passive
a=dtls-id:1
a=rtcp-mux
```

```
a=rtcp-mux-only
a=rtcp-rsize
a=candidate:1 1 udp 255 192.0.2.100 12100 typ relay
            raddr 0.0.0.0 rport 0
a=end-of-candidates

m=video 12100 UDP/TLS/RTP/SAVPF 100 101
c=IN IP4 192.0.2.100
a=mid:v1
a=sendrecv
a=rtpmap:100 VP8/90000
a=rtpmap:101 rtx/90000
a=fmtp:101 apt=100
a=extmap:1 urn:ietf:params:rtp-hdrext:sdes:mid
a=rtcp-fb:100 ccm fir
a=rtcp-fb:100 nack
a=rtcp-fb:100 nack pli
a=msid:bbce3ba6-abfc-ac63-d00a-e15b286f8fce
       ac701365-eb06-42df-cc93-7f22bc308789
```

8. Security Considerations

The IETF has published separate documents [I-D.ietf-rtcweb-security-arch] [I-D.ietf-rtcweb-security] describing the security architecture for WebRTC as a whole. The remainder of this section describes security considerations for this document.

While formally the JSEP interface is an API, it is better to think of it is an Internet protocol, with the JS being untrustworthy from the perspective of the endpoint. Thus, the threat model of [RFC3552] applies. In particular, JS can call the API in any order and with any inputs, including malicious ones. This is particularly relevant when we consider the SDP which is passed to setLocalDescription(). While correct API usage requires that the application pass in SDP which was derived from createOffer() or createAnswer(), there is no guarantee that applications do so. The JSEP implementation MUST be prepared for the JS to pass in bogus data instead.

Conversely, the application programmer MUST recognize that the JS does not have complete control of endpoint behavior. One case that bears particular mention is that editing ICE candidates out of the SDP or suppressing trickled candidates does not have the expected behavior: implementations will still perform checks from those candidates even if they are not sent to the other side. Thus, for instance, it is not possible to prevent the remote peer from learning your public IP address by removing server reflexive candidates.

Applications which wish to conceal their public IP address should instead configure the ICE agent to use only relay candidates.

9. IANA Considerations

This document requires no actions from IANA.

10. Acknowledgements

Harald Alvestrand, Taylor Brandstetter, Suhas Nandakumar, and Peter Thatcher provided significant text for this draft. Bernard Aboba, Adam Bergkvist, Dan Burnett, Ben Campbell, Alissa Cooper, Richard Ejzak, Stefan Hakansson, Ted Hardie, Christer Holmberg Andrew Hutton, Randell Jesup, Matthew Kaufman, Anant Narayanan, Adam Roach, Neil Stratford, Martin Thomson, Sean Turner, and Magnus Westerlund all provided valuable feedback on this proposal.

11. References

11.1. Normative References

[I-D.ietf-avtcore-rtp-multi-stream]
Lennox, J., Westerlund, M., Wu, Q., and C. Perkins, "Sending Multiple RTP Streams in a Single RTP Session", draft-ietf-avtcore-rtp-multi-stream-11 (work in progress), December 2015.

[I-D.ietf-avtext-rid]
Roach, A., Nandakumar, S., and P. Thatcher, "RTP Stream Identifier (RID) Source Description (SDES)", draft-ietf-avtext-rid-00 (work in progress), February 2016.

[I-D.ietf-ice-trickle]
Ivov, E., Rescorla, E., Uberti, J., and P. Saint-Andre, "Trickle ICE: Incremental Provisioning of Candidates for the Interactive Connectivity Establishment (ICE) Protocol".

[I-D.ietf-mmusic-4572-update]
Holmberg, C., "Updates to RFC 4572", draft-ietf-mmusic-4572-update-05 (work in progress), June 2016.

[I-D.ietf-mmusic-dtls-sdp]
Holmberg, C. and R. Shpount, "Using the SDP Offer/Answer Mechanism for DTLS", draft-ietf-mmusic-dtls-sdp-14 (work in progress), July 2016.

[I-D.ietf-mmusic-msid]
Alvestrand, H., "Cross Session Stream Identification in the Session Description Protocol", draft-ietf-mmusic-msid-01 (work in progress), August 2013.

[I-D.ietf-mmusic-mux-exclusive]
Holmberg, C., "Indicating Exclusive Support of RTP/RTCP Multiplexing using SDP", draft-ietf-mmusic-mux-exclusive-08 (work in progress), June 2016.

[I-D.ietf-mmusic-rid]
Thatcher, P., Zanaty, M., Nandakumar, S., Burman, B., Roach, A., and B. Campen, "RTP Payload Format Constraints", draft-ietf-mmusic-rid-04 (work in progress), February 2016.

[I-D.ietf-mmusic-sctp-sdp]
Loreto, S. and G. Camarillo, "Stream Control Transmission Protocol (SCTP)-Based Media Transport in the Session Description Protocol (SDP)", draft-ietf-mmusic-sctp-sdp-04 (work in progress), June 2013.

[I-D.ietf-mmusic-sdp-bundle-negotiation]
Holmberg, C., Alvestrand, H., and C. Jennings, "Multiplexing Negotiation Using Session Description Protocol (SDP) Port Numbers", draft-ietf-mmusic-sdp-bundle-negotiation-04 (work in progress), June 2013.

[I-D.ietf-mmusic-sdp-mux-attributes]
Nandakumar, S., "A Framework for SDP Attributes when Multiplexing", draft-ietf-mmusic-sdp-mux-attributes-01 (work in progress), February 2014.

[I-D.ietf-mmusic-sdp-simulcast]
Burman, B., Westerlund, M., Nandakumar, S., and M. Zanaty, "Using Simulcast in SDP and RTP Sessions", draft-ietf-mmusic-sdp-simulcast-04 (work in progress), February 2016.

[I-D.ietf-rtcweb-audio]
Valin, J. and C. Bran, "WebRTC Audio Codec and Processing Requirements", draft-ietf-rtcweb-audio-02 (work in progress), August 2013.

[I-D.ietf-rtcweb-fec]
Uberti, J., "WebRTC Forward Error Correction Requirements", draft-ietf-rtcweb-fec-00 (work in progress), February 2015.

[I-D.ietf-rtcweb-rtp-usage]
Perkins, C., Westerlund, M., and J. Ott, "Web Real-Time Communication (WebRTC): Media Transport and Use of RTP", draft-ietf-rtcweb-rtp-usage-09 (work in progress), September 2013.

[I-D.ietf-rtcweb-security]
Rescorla, E., "Security Considerations for WebRTC", draft-ietf-rtcweb-security-06 (work in progress), January 2014.

[I-D.ietf-rtcweb-security-arch]
Rescorla, E., "WebRTC Security Architecture", draft-ietf-rtcweb-security-arch-09 (work in progress), February 2014.

[I-D.ietf-rtcweb-video]
Roach, A., "WebRTC Video Processing and Codec Requirements", draft-ietf-rtcweb-video-00 (work in progress), July 2014.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.

[RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, July 2003.

[RFC3605] Huitema, C., "Real Time Control Protocol (RTCP) attribute in Session Description Protocol (SDP)", RFC 3605, October 2003.

[RFC3890] Westerlund, M., "A Transport Independent Bandwidth Modifier for the Session Description Protocol (SDP)", RFC 3890, DOI 10.17487/RFC3890, September 2004, <http://www.rfc-editor.org/info/rfc3890>.

[RFC4145] Yon, D. and G. Camarillo, "TCP-Based Media Transport in the Session Description Protocol (SDP)", RFC 4145, September 2005.

[RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, July 2006.

[RFC4572] Lennox, J., "Connection-Oriented Media Transport over the Transport Layer Security (TLS) Protocol in the Session Description Protocol (SDP)", RFC 4572, July 2006.

[RFC4585] Ott, J., Wenger, S., Sato, N., Burmeister, C., and J. Rey, "Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/AVPF)", RFC 4585, July 2006.

[RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.

[RFC5285] Singer, D. and H. Desineni, "A General Mechanism for RTP Header Extensions", RFC 5285, July 2008.

[RFC5761] Perkins, C. and M. Westerlund, "Multiplexing RTP Data and Control Packets on a Single Port", RFC 5761, April 2010.

[RFC5888] Camarillo, G. and H. Schulzrinne, "The Session Description Protocol (SDP) Grouping Framework", RFC 5888, June 2010.

[RFC6236] Johansson, I. and K. Jung, "Negotiation of Generic Image Attributes in the Session Description Protocol (SDP)", RFC 6236, May 2011.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.

[RFC6716] Valin, JM., Vos, K., and T. Terriberry, "Definition of the Opus Audio Codec", RFC 6716, DOI 10.17487/RFC6716, September 2012, <http://www.rfc-editor.org/info/rfc6716>.

[RFC6904] Lennox, J., "Encryption of Header Extensions in the Secure Real-time Transport Protocol (SRTP)", RFC 6904, April 2013.

[RFC7160] Petit-Huguenin, M. and G. Zorn, Ed., "Support for Multiple Clock Rates in an RTP Session", RFC 7160, DOI 10.17487/RFC7160, April 2014, <http://www.rfc-editor.org/info/rfc7160>.

[RFC7587] Spittka, J., Vos, K., and JM. Valin, "RTP Payload Format for the Opus Speech and Audio Codec", RFC 7587, DOI 10.17487/RFC7587, June 2015, <http://www.rfc-editor.org/info/rfc7587>.

[RFC7850] Nandakumar, S., "Registering Values of the SDP 'proto' Field for Transporting RTP Media over TCP under Various RTP Profiles", RFC 7850, DOI 10.17487/RFC7850, April 2016, <http://www.rfc-editor.org/info/rfc7850>.

[RFC7941] Westerlund, M., Burman, B., Even, R., and M. Zanaty, "RTP Header Extension for the RTP Control Protocol (RTCP) Source Description Items", RFC 7941, DOI 10.17487/RFC7941, August 2016, <http://www.rfc-editor.org/info/rfc7941>.

11.2. Informative References

[I-D.ietf-avtext-lrr] Lennox, J., Hong, D., Uberti, J., Homer, S., and M. Flodman, "The Layer Refresh Request (LRR) RTCP Feedback Message", draft-ietf-avtext-lrr-03 (work in progress), July 2016.

[I-D.ietf-rtcweb-ip-handling] Uberti, J. and G. Shieh, "WebRTC IP Address Handling Recommendations", draft-ietf-rtcweb-ip-handling-01 (work in progress), March 2016.

[I-D.nandakumar-rtcweb-sdp] Nandakumar, S. and C. Jennings, "SDP for the WebRTC", draft-nandakumar-rtcweb-sdp-02 (work in progress), July 2013.

[RFC3389] Zopf, R., "Real-time Transport Protocol (RTP) Payload for Comfort Noise (CN)", RFC 3389, September 2002.

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, DOI 10.17487/RFC3550, July 2003, <http://www.rfc-editor.org/info/rfc3550>.

[RFC3556] Casner, S., "Session Description Protocol (SDP) Bandwidth Modifiers for RTP Control Protocol (RTCP) Bandwidth", RFC 3556, July 2003.

[RFC3611] Friedman, T., Caceres, R., and A. Clark, "RTP Control Protocol Extended Reports (RTCP XR)", RFC 3611, DOI 10.17487/RFC3611, November 2003, <http://www.rfc-editor.org/info/rfc3611>.

[RFC3960] Camarillo, G. and H. Schulzrinne, "Early Media and Ringing Tone Generation in the Session Initiation Protocol (SIP)", RFC 3960, December 2004.

[RFC4568] Andreasen, F., Baugher, M., and D. Wing, "Session Description Protocol (SDP) Security Descriptions for Media Streams", RFC 4568, July 2006.

[RFC4588] Rey, J., Leon, D., Miyazaki, A., Varsa, V., and R. Hakenberg, "RTP Retransmission Payload Format", RFC 4588, July 2006.

[RFC4733] Schulzrinne, H. and T. Taylor, "RTP Payload for DTMF Digits, Telephony Tones, and Telephony Signals", RFC 4733, DOI 10.17487/RFC4733, December 2006, <http://www.rfc-editor.org/info/rfc4733>.

[RFC5104] Wenger, S., Chandra, U., Westerlund, M., and B. Burman, "Codec Control Messages in the RTP Audio-Visual Profile with Feedback (AVPF)", RFC 5104, DOI 10.17487/RFC5104, February 2008, <http://www.rfc-editor.org/info/rfc5104>.

[RFC5506] Johansson, I. and M. Westerlund, "Support for Reduced-Size Real-Time Transport Control Protocol (RTCP): Opportunities and Consequences", RFC 5506, April 2009.

[RFC5576] Lennox, J., Ott, J., and T. Schierl, "Source-Specific Media Attributes in the Session Description Protocol (SDP)", RFC 5576, June 2009.

[RFC5763] Fischl, J., Tschofenig, H., and E. Rescorla, "Framework for Establishing a Secure Real-time Transport Protocol (SRTP) Security Context Using Datagram Transport Layer Security (DTLS)", RFC 5763, May 2010.

[RFC5764] McGrew, D. and E. Rescorla, "Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP)", RFC 5764, May 2010.

[RFC6464] Lennox, J., Ed., Ivov, E., and E. Marocco, "A Real-time Transport Protocol (RTP) Header Extension for Client-to-Mixer Audio Level Indication", RFC 6464, DOI 10.17487/RFC6464, December 2011, <http://www.rfc-editor.org/info/rfc6464>.

[RFC6544] Rosenberg, J., Keranen, A., Lowekamp, B., and A. Roach, "TCP Candidates with Interactive Connectivity Establishment (ICE)", RFC 6544, DOI 10.17487/RFC6544, March 2012, <http://www.rfc-editor.org/info/rfc6544>.

[RFC7656] Lennox, J., Gross, K., Nandakumar, S., Salgueiro, G., and B. Burman, Ed., "A Taxonomy of Semantics and Mechanisms for Real-Time Transport Protocol (RTP) Sources", RFC 7656, DOI 10.17487/RFC7656, November 2015, <http://www.rfc-editor.org/info/rfc7656>.

[TS26.114] 3GPP TS 26.114 V12.8.0, "3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; IP Multimedia Subsystem (IMS); Multimedia Telephony; Media handling and interaction (Release 12)", December 2014, <http://www.3gpp.org/DynaReport/26114.htm>.

[W3C.WD-webrtc-20140617] Bergkvist, A., Burnett, D., Narayanan, A., and C. Jennings, "WebRTC 1.0: Real-time Communication Between Browsers", World Wide Web Consortium WD WD-webrtc-20140617, June 2014, <http://www.w3.org/TR/2011/WD-webrtc-20140617>.

Appendix A. Appendix A

For the syntax validation performed in Section 5.7, the following list of ABNF definitions is used:

Attribute	Reference
ptime	[RFC4566] Section 9
maxptime	[RFC4566] Section 9
rtpmap	[RFC4566] Section 9
recvonly	[RFC4566] Section 9
sendrecv	[RFC4566] Section 9
sendonly	[RFC4566] Section 9
inactive	[RFC4566] Section 9
framerate	[RFC4566] Section 9
fmtp	[RFC4566] Section 9
quality	[RFC4566] Section 9
rtcp	[RFC3605] Section 2.1
setup	[RFC4145] Sections 3, 4, and 5
connection	[RFC4145] Sections 3, 4, and 5
fingerprint	[RFC4572] Section 5
rtcp-fb	[RFC4585] Section 4.2
candidate	[RFC5245] Section 15.1
remote-candidates	[RFC5245] Section 15.2
ice-lite	[RFC5245] Section 15.3
ice-ufrag	[RFC5245] Section 15.4
ice-pwd	[RFC5245] Section 15.4
ice-options	[RFC5245] Section 15.5
extmap	[RFC5285] Section 7
mid	[RFC5888] Section 4 and 5
group	[RFC5888] Section 4 and 5
imageattr	[RFC6236] Section 3.1
extmap (encrypt option)	[RFC6904] Section 4
msid	[I-D.ietf-mmusic-msid] Section 2
rid	[I-D.ietf-mmusic-rid] Section 10
simulcast	[I-D.ietf-mmusic-sdp-simulcast]Section 6.1
dtls-id	[I-D.ietf-mmusic-dtls-sdp]Section 4

Table 1: SDP ABNF References

Appendix B. Appendix B

The following text is meant to completely replace section "Associating RTP/RTCP Streams With Correct SDP Media Description" of [I-D.ietf-mmusic-sdp-bundle-negotiation].

As described in [RFC3550], RTP packets are associated with RTP streams [RFC7656]. Each RTP stream is identified by an SSRC value, and each RTP packet includes an SSRC field that is used to associate

the packet with the correct RTP stream. RTCP packets also use SSRCs to identify which RTP streams the packet relates to. However, a RTCP packet can contain multiple SSRC fields, in the course of providing feedback or reports on different RTP streams, and therefore can be associated with multiple such streams.

In order to be able to process received RTP/RTCP packets correctly, it must be possible to associate an RTP stream with the correct "m=" line, as the "m=" line and SDP attributes associated with the "m=" line contain information needed to process the packets.

As all RTP streams associated with a BUNDLE group use the same address:port combination for sending and receiving RTP/RTCP packets, the local address:port combination cannot be used to associate an RTP stream with the correct "m=" line. In addition, multiple RTP streams might be associated with the same "m=" line.

An offerer and answerer can inform each other which SSRC values they will use for an RTP stream by using the SDP 'ssrc' attribute [RFC5576]. However, an offerer will not know which SSRC values the answerer will use until the offerer has received the answer providing that information. Due to this, before the offerer has received the answer, the offerer will not be able to associate an RTP stream with the correct "m=" line using the SSRC value associated with the RTP stream. In addition, the offerer and answerer may start using new SSRC values mid-session, without informing each other using the SDP 'ssrc' attribute.

In order for an offerer and answerer to always be able to associate an RTP stream with the correct "m=" line, the offerer and answerer using the BUNDLE extension MUST support the mechanism defined in [I-D.ietf-mmusic-sdp-bundle-negotiation], Section 14, where the offerer and answerer insert the identification-tag associated with an "m=" line (provided by the remote peer) into RTP and RTCP packets associated with a BUNDLE group.

When using this mechanism, the mapping from an SSRC to an identification-tag is carried in RTP header extensions or RTCP SDES packets, as specified in [I-D.ietf-mmusic-sdp-bundle-negotiation], Section 14). Since a compound RTCP packet can contain multiple RTCP SDES packets, and each RTCP SDES packet can contain multiple chunks, a single RTCP packet can contain several SSRC to identification-tag mappings. The offerer and answerer maintain tables used for routing that are updated each time an RTP/RTCP packet contains new information that affects how packets should be routed.

However, some implementations of [I-D.ietf-mmusic-sdp-bundle-negotiation] may not include this

identification-tag in their RTP and RTCP traffic when using BUNDLE, and instead use a payload type based mechanism for demuxing. In this situation, each "m=" line MUST use unique payload type values, in order for the payload type to be a reliable indicator of the relevant "m=" line for the RTP stream.

Applications can implement RTP stacks in many different ways. The algorithm below details one way that demultiplexing can be accomplished, but is not meant to be prescriptive about exactly how an RTP stack needs to be implemented. Applications MAY use any algorithm that achieves equivalent results to those described in the algorithm below.

To prepare for demultiplexing RTP/RTCP packets to the correct "m=" line, the following steps MUST be followed for each BUNDLE group.

> Construct a table mapping MID to "m=" line for each "m=" line in this BUNDLE group. Note that an "m=" line may only have one MID.
>
> Construct a table mapping incoming SSRC to "m=" line for each "m=" line in this BUNDLE group and for each SSRC configured for receiving in that "m=" line.
>
> Construct a table mapping outgoing SSRC to "m=line" for each "m=" line in this BUNDLE group and for each SSRC configured for sending in that "m=" line.
>
> Construct a table mapping payload type to "m=" line for each "m=" line in the BUNDLE group and for each payload type configured for receiving in that "m=" line. If any payload type is configured for receiving in more than one "m=" line in the BUNDLE group, do not it include it in the table, as it cannot be used to uniquely identify a "m=" line.
>
> Note that for each of these tables, there can only be one mapping for any given key (MID, SSRC, or PT). In other words, the tables are not multimaps.

As "m=" lines are added or removed from the BUNDLE groups, or their configurations are changed, the tables above MUST also be updated.

For each RTP packet received, the following steps MUST be followed to route the packet to the correct "m=" section within a BUNDLE group. Note that the phrase 'deliver a packet to the "m=" line' means to further process the packet as would normally happen with RTP/RTCP, if it were received on a transport associated with that "m=" line outside of a BUNDLE group (i.e., if the "m=" line were not BUNDLEd),

including dropping an RTP packet if the packet's PT does not match any PT in the "m=" line.

If the packet has a MID, and that MID is not in the table mapping MID to "m=" line, drop the packet and stop.

If the packet has a MID, and the packet's extended sequence number is greater than that of the last MID update, as discussed in [RFC7941], Section 4.2.6, update the incoming SSRC mapping table to include an entry that maps the packet's SSRC to the "m=" line for that MID.

If the packet's SSRC is in the incoming SSRC mapping table, check that the packet's PT matches a PT included on the associated "m=" line. If so, route the packet to that associated "m=" line and stop; otherwise drop the packet and stop.

If the packet's payload type is in the payload type table, update the the incoming SSRC mapping table to include an entry that maps the packet's SSRC to the "m=" line for that payload type. In addition, route the packet to the associated "m=" line and stop.

Otherwise, drop the packet.

For each RTCP packet received (including each RTCP packet that is part of a compound RTCP packet), the packet MUST be routed to the "m=" line for the RTP streams it contains information about. This routing is type-dependent, as each kind of RTCP packet has its own mechanism for associating it with the relevant RTP streams.

Packets for which no appropriate "m=" line can be identified (i.e., for unknown RTP streams) are not relevant in the context of this algorithm and MAY be dropped. This situation may occur with certain multiparty RTP topologies.

Rules for handling the various types of RTCP packets are explained below.

If the packet is of type SDES, for each chunk in the packet whose SSRC is found in the incoming SSRC table, deliver a copy of the packet to the "m=" line associated with that SSRC. In addition, for any SDES MID items contained in these chunks, if the MID is found in the table mapping MID to "m=" line, update the incoming SSRC table to include an entry that maps the chunk's SSRC to the "m=" line associated with that MID, unless the packet is older than the packet that most recently updated the mapping for this SSRC, as discussed in [RFC7941], Section 4.2.6.

Note that if an SDES packet is received as part of a compound RTCP packet, the SSRC to "m=" line mapping may not exist until the SDES packet is handled (e.g., in the case where RTCP for a source is received before any RTP packets). Therefore, when processing a compound packet, any contained SDES packet MUST be handled first.

If the packet is of type BYE, it indicates that the RTP streams referenced in the packet are ending. Therefore, for each SSRC indicated in the packet that is found in the incoming SSRC table, first deliver a copy of the packet to the "m=" line associated with that SSRC, but then remove the entry for that SSRC from the incoming SSRC table.

If the packet is of type SR or RR, for each report block in the report whose "SSRC of source" is found in the outgoing SSRC table, deliver a copy of the RTCP packet to the "m=" line associated with that SSRC. In addition, if the packet is of type SR, and the sender SSRC for the packet is found in the incoming SSRC table, deliver a copy of the packet to the "m=" line associated with that SSRC.

If the implementation supports RTCP XR and the packet is of type XR, as defined in [RFC3611], for each report block in the report whose "SSRC of source" is is found in the outgoing SSRC table, deliver a copy of the RTCP packet to the "m=" line associated with that SSRC. In addition, if the sender SSRC for the packet is found in the incoming SSRC table, deliver a copy of the packet to the "m=" line associated with that SSRC.

If the packet is a feedback message of type RTPFB or PSFB, as defined in [RFC4585], it will contain a media source SSRC, and this SSRC is used for routing certain subtypes of feedback messages. However, several subtypes of PSFB messages include target SSRC(s) in a section called Feedback Control Information (FCI). For these messages, the target SSRC(s) are used for routing.

If the packet is a feedback message that does not include target SSRCs in its FCI section, and the media source SSRC is found in the outgoing SSRC table, deliver the packet to the "m=" line associated with that SSRC. RTPFB and PSFB types that are handled in this way include:

Generic NACK: [RFC4585] (PT=RTPFB, FMT=1).

Picture Loss Indication (PLI): [RFC4585] (PT=PSFB, FMT=1).

Slice Loss Indication (SLI): [RFC4585] (PT=PSFB, FMT=2).

Reference Picture Selection Indication (RPSI): [RFC4585] (PT=PSFB, FMT=3).

If the packet is a feedback message that does include target SSRC(s) in its FCI section, it can either be a request or a notification. Requests reference a RTP stream that is being sent by the message recipient, whereas notifications are responses to an earlier request, and therefore reference a RTP stream that is being received by the message recipient.

If the packet is a feedback request that includes target SSRC(s), for each target SSRC that is found in the outgoing SSRC table, deliver a copy of the RTCP packet to the "m=" line associated with that SSRC. PSFB types that are handled in this way include:

Full Intra Request (FIR): [RFC5104] (PT=PSFB, FMT=4).

Temporal-Spatial Trade-off Request (TSTR): [RFC5104] (PT=PSFB, FMT=5).

H.271 Video Back Channel Message (VBCM): [RFC5104] (PT=PSFB, FMT=7).

Layer Refresh Request (LRR): [I-D.ietf-avtext-lrr] (PT=PSFB, FMT=TBD).

If the packet is a feedback notification that include target SSRC(s), for each target SSRC that is found in the incoming SSRC table, deliver a copy of the RTCP packet to the "m=" line associated with that SSRC. PSFB types that are handled in this way include:

Temporal-Spatial Trade-off Notification (TSTN): [RF C5104] (PT=PSFB, FMT=6). This message is a notification in response to a prior TSTR.

If the packet is of type APP, the only routing information included is the source of the packet, and therefore the packet could be related to any existing "m=" line. Accordingly, deliver a copy of the packet to each "m=" line.

Appendix C. Change log

Note: This section will be removed by RFC Editor before publication.

Changes in draft-19:

o Examples are now machine-generated for correctness, and use IETF-approved example IP addresses.

o Add early transport warmup example, and add missing attributes to existing examples.

o Only send "a=rtcp-mux-only" and "a=bundle-only" on new m= sections.

o Update references.

o Add coverage of a=identity.

o Explain the lipsync group algorithm more thoroughly.

o Remove unnecessary list of MTI specs.

o Allow codecs which weren't offered to appear in answers and which weren't selected to appear in subsequent offers.

o Codec preferences now are applied on both initial and subsequent offers and answers.

o Clarify a=msid handling for recvonly m= sections.

o Clarify behavior of attributes for bundle-only data channels.

o Allow media attributes to appear in data m= sections when all the media m= sections are bundle-only.

o Use consistent terminology for JSEP implementations.

o Describe how to handle failed API calls.

o Some cleanup on routing rules.

Changes in draft-18:

o Update demux algorithm and move it to an appendix in preparation for merging it into BUNDLE.

o Clarify why we can't handle an incoming offer to send simulcast.

o Expand IceCandidate object text.

o Further document use of ICE candidate pool.

o Document removeTrack.

o Update requirements to only accept the last generated offer/answer as an argument to setLocalDescription.

o Allow round pixels.

o Fix code around default timing when AVPF is not specified.

o Clean up terminology around m= line and m=section.

o Provide a more realistic example for minimum decoder capabilities.

o Document behavior when rtcp-mux policy is require but rtcp-mux attribute not provided.

o Expanded discussion of RtpSender and RtpReceiver.

o Add RtpTransceiver.currentDirection and document setDirection.

o Require imageattr x=0, y=0 to indicate that there are no valid resolutions.

o Require a privacy-preserving MID/RID construction.

o Require support for RFC 3556 bandwidth modifiers.

o Update maxptime description.

o Note that endpoints may encounter extra codecs in answers and subsequent offers from non-JSEP peers.

o Update references.

Changes in draft-17:

o Split createOffer and createAnswer sections to clearly indicate attributes which always appear and which only appear when not bundled into another m= section.

o Add descriptions of RtpTransceiver methods.

o Describe how to process RTCP feedback attributes.

o Clarify transceiver directions and their interaction with 3264.

o Describe setCodecPreferences.

o Update RTP demux algorithm. Include RTCP.

- o Update requirements for when a=rtcp is included, limiting to cases where it is needed for backward compatibility.

- o Clarify SAR handling.

- o Updated addTrack matching algorithm.

- o Remove a=ssrc requirements.

- o Handle a=setup in reoffers.

- o Discuss how RTX/FEC should be handled.

- o Discuss how telephone-event should be handled.

- o Discuss how CN/DTX should be handled.

- o Add missing references to ABNF table.

Changes in draft-16:

- o Update addIceCandidate to indicate ICE generation and allow per-m= section end-of-candidates.

- o Update fingerprint handling to use draft-ietf-mmusic-4572-update.

- o Update text around SDP processing of RTP header extensions and payload formats.

- o Add sections on simulcast, addTransceiver, and createDataChannel.

- o Clarify text to ensure that the session ID is a positive 63 bit integer.

- o Clarify SDP processing for direction indication.

- o Describe SDP processing for rtcp-mux-only.

- o Specify how SDP session version in o= line.

- o Require that when doing an re-offer, the capabilities of the new session are mostly required to be a subset of the previously negotiated session.

- o Clarified ICE restart interaction with bundle-only.

- o Remove support for changing SDP before calling setLocalDescription.

- Specify algorithm for demuxing RTP based on MID, PT, and SSRC.

- Clarify rules for rejecting m= lines when bundle policy is balanced or max-bundle.

Changes in draft-15:

- Clarify text around codecs offered in subsequent transactions to refer to what's been negotiated.

- Rewrite LS handling text to indicate edge cases and that we're living with them.

- Require that answerer reject m= lines when there are no codecs in common.

- Enforce max-bundle on offer processing.

- Fix TIAS formula to handle bits vs. kilobits.

- Describe addTrack algorithm.

- Clean up references.

Changes in draft-14:

- Added discussion of RtpTransceivers + RtpSenders + RtpReceivers, and how they interact with createOffer/createAnswer.

- Removed obsolete OfferToReceiveX options.

- Explained how addIceCandidate can be used for end-of-candidates.

Changes in draft-13:

- Clarified which SDP lines can be ignored.

- Clarified how to handle various received attributes.

- Revised how attributes should be generated for bundled m= lines.

- Remove unused references.

- Remove text advocating use of unilateral PTs.

- Trigger an ICE restart even if the ICE candidate policy is being made more strict.

o Remove the 'public' ICE candidate policy.

o Move open issues into GitHub issues.

o Split local/remote description accessors into current/pending.

o Clarify a=imageattr handling.

o Add more detail on VoiceActivityDetection handling.

o Reference draft-shieh-rtcweb-ip-handling.

o Make it clear when an ICE restart should occur.

o Resolve changes needed in references.

o Remove MSID semantics.

o ice-options are now at session level.

o Default RTCP mux policy is now 'require'.

Changes in draft-12:

o Filled in sections on applying local and remote descriptions.

o Discussed downscaling and upscaling to fulfill imageattr requirements.

o Updated what SDP can be modified by the application.

o Updated to latest datachannel SDP.

o Allowed multiple fingerprint lines.

o Switched back to IPv4 for dummy candidates.

o Added additional clarity on ICE default candidates.

Changes in draft-11:

o Clarified handling of RTP CNAMEs.

o Updated what SDP lines should be processed or ignored.

o Specified how a=imageattr should be used.

Changes in draft-10:

o Described video size negotiation with imageattr.

o Clarified rejection of sections that do not have mux-only.

o Add handling of LS groups

Changes in draft-09:

o Don't return null for {local,remote}Description after close().

o Changed TCP/TLS to UDP/DTLS in RTP profile names.

o Separate out bundle and mux policy.

o Added specific references to FEC mechanisms.

o Added canTrickle mechanism.

o Added section on subsequent answers and, answer options.

o Added text defining set{Local,Remote}Description behavior.

Changes in draft-08:

o Added new example section and removed old examples in appendix.

o Fixed <proto> field handling.

o Added text describing a=rtcp attribute.

o Reworked handling of OfferToReceiveAudio and OfferToReceiveVideo per discussion at IETF 90.

o Reworked trickle ICE handling and its impact on m= and c= lines per discussion at interim.

o Added max-bundle-and-rtcp-mux policy.

o Added description of maxptime handling.

o Updated ICE candidate pool default to 0.

o Resolved open issues around AppID/receiver-ID.

o Reworked and expanded how changes to the ICE configuration are handled.

o Some reference updates.

- o Editorial clarification.

Changes in draft-07:

- o Expanded discussion of VAD and Opus DTX.
- o Added a security considerations section.
- o Rewrote the section on modifying SDP to require implementations to clearly indicate whether any given modification is allowed.
- o Clarified impact of IceRestart on CreateOffer in local-offer state.
- o Guidance on whether attributes should be defined at the media level or the session level.
- o Renamed "default" bundle policy to "balanced".
- o Removed default ICE candidate pool size and clarify how it works.
- o Defined a canonical order for assignment of MSTs to m= lines.
- o Removed discussion of rehydration.
- o Added Eric Rescorla as a draft editor.
- o Cleaned up references.
- o Editorial cleanup

Changes in draft-06:

- o Reworked handling of m= line recycling.
- o Added handling of BUNDLE and bundle-only.
- o Clarified handling of rollback.
- o Added text describing the ICE Candidate Pool and its behavior.
- o Allowed OfferToReceiveX to create multiple recvonly m= sections.

Changes in draft-05:

- o Fixed several issues identified in the createOffer/Answer sections during document review.

o Updated references.

Changes in draft-04:

o Filled in sections on createOffer and createAnswer.

o Added SDP examples.

o Fixed references.

Changes in draft-03:

o Added text describing relationship to W3C specification

Changes in draft-02:

o Converted from nroff

o Removed comparisons to old approaches abandoned by the working group

o Removed stuff that has moved to W3C specification

o Align SDP handling with W3C draft

o Clarified section on forking.

Changes in draft-01:

o Added diagrams for architecture and state machine.

o Added sections on forking and rehydration.

o Clarified meaning of "pranswer" and "answer".

o Reworked how ICE restarts and media directions are controlled.

o Added list of parameters that can be changed in a description.

o Updated suggested API and examples to match latest thinking.

o Suggested API and examples have been moved to an appendix.

Changes in draft -00:

o Migrated from draft-uberti-rtcweb-jsep-02.

Authors' Addresses

Justin Uberti
Google
747 6th St S
Kirkland, WA 98033
USA

Email: justin@uberti.name

Cullen Jennings
Cisco
400 3rd Avenue SW
Calgary, AB T2P 4H2
Canada

Email: fluffy@iii.ca

Eric Rescorla (editor)
Mozilla
331 Evelyn Ave
Mountain View, CA 94041
USA

Email: ekr@rtfm.com