Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 10, 2015

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

March 9, 2015

Javascript Session Establishment Protocol
draft-ietf-rtcweb-jsep-09

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

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 10, 2015.

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. JSEP removes the browser almost entirely 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.

In this document, the use of JSEP is described as if it always occurs between two browsers. Note though in many cases it will actually be between a browser and some kind of server, such as a gateway or MCU. This distinction is invisible to the browser; 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 optionally modifies that offer, and then uses it to set up its local config via the setLocalDescription() API. The offer is then 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 browser, because only the browser 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 browser, forcing the browser 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 browser. Instead of providing createOffer/createAnswer methods within the browser, 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, but still allows applications to generate their own offers/answers (to a large extent) if they choose, using the description generated by createOffer as an indication of the browser's capabilities.

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 SDP media 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 browser 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 browser, and when.

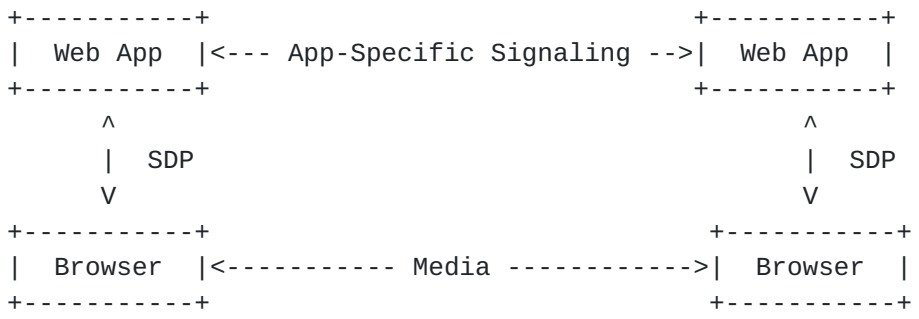

Figure 1: JSEP Signaling Model

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 DTLS-SRTP parameters [RFC5763] sent to a remote party indicate what certificate the local side will use in DTLS setup, and thereby what the remote party should expect to receive; the remote party will have to accept these parameters, 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 media streams (i.e. m= sections), 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 results; as a result, multiple dissimilar provisional answers can be received and applied during call setup.

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:

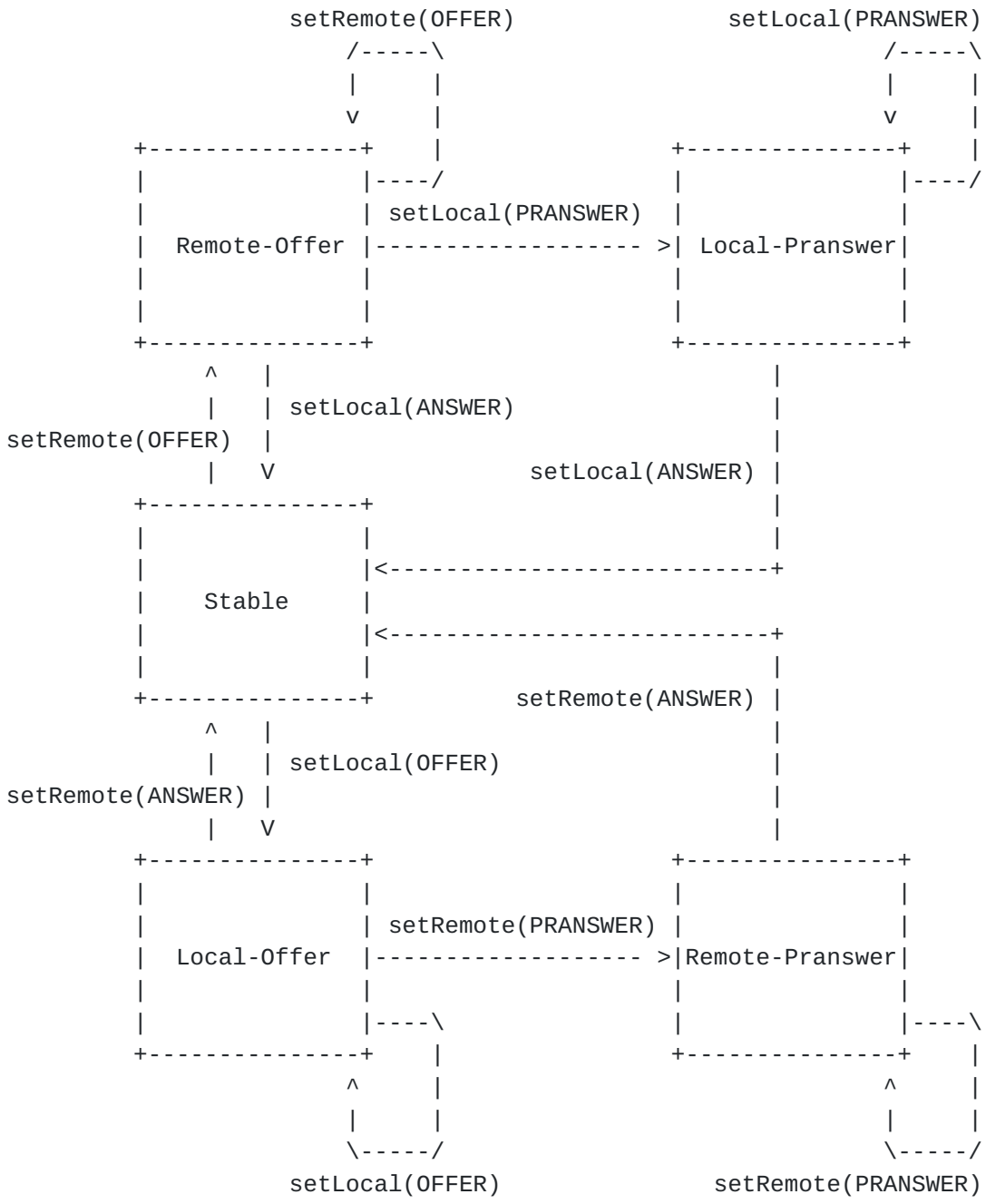

Figure 2: JSEP State Machine

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

In the WebRTC specification, session descriptions are formatted as SDP messages. 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. As a result, JSEP currently uses SDP as the internal representation for its session descriptions.

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. The W3C WebRTC API specification will provide appropriate APIs to allow the application to control various session parameters, which will provide the necessary information to the browser about what sort of SessionDescription to produce.

3.4. ICE

3.4.1. 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= line 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 browser in response to changes in the ICE configuration.

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 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= lines; other m= lines continue to use their existing candidates.

3.4.2. 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-mmusic-trickle-ice]. 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.

3.4.2.1. ICE Candidate Format

As with session descriptions, the syntax of the IceCandidate object provides some abstraction, but can be easily converted to and from the SDP candidate lines.

The candidate lines are the only SDP information that is contained within IceCandidate, as they represent the only information needed that is not present in the initial offer (i.e., for trickle candidates). This information is carried with the same syntax as the "candidate-attribute" field defined for ICE. For example:

candidate:1 1 UDP 1694498815 192.0.2.33 10000 typ host

The IceCandidate object also contains fields to indicate which m= line it should be associated with. The m= line can be identified in

one of two ways; either by a m= line index, or a MID. The m= line index is a zero-based index, with index N referring to the N+1th m= line in the SDP sent by the entity which sent the IceCandidate. The MID uses the "media stream identification" attribute, as defined in [RFC5888], Section 4, to identify the m= line. JSEP implementations creating an ICE Candidate object MUST populate both of these fields. Implementations receiving an ICE Candidate object MUST use the MID if present, or the m= line index, if not (as it could have come from a non-JSEP endpoint).

3.4.3. ICE Candidate Policy

Typically, when gathering ICE candidates, the browser 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 suppress the use of host candidates, to avoid exposing information about the local network, or go as far as only using relay candidates, to leak as little location information as possible (note that these choices come with corresponding operational costs). To accomplish this, the browser MUST allow the application to restrict which ICE candidates are used in a session. In addition, administrators may also wish to control the set of ICE candidates, and so the browser SHOULD also allow control via local policy, with the most restrictive policy prevailing.

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 browser 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 browser will determine the current setting at the start of each gathering phase. Then, during the gathering phase, the browser 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.

3.4.4. ICE Candidate Pool

JSEP applications typically inform the browser 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 browser to gather a pool of potential ICE candidates to help ensure rapid media setup.

When setLocalDescription is eventually called, and the browser 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.

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.

3.5. 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:

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

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

3.5.1. 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 existing remote description as a final answer.

3.5.2. 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 by specifying SDP direction attributes in the descriptions, 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. Methods

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 3.4.3, causing the browser to only surface the permitted candidates to the application, and only use those candidates for connectivity checks. The set of available policies is as follows:

all: All candidates will be gathered and used.

public: Candidates with private IP addresses [RFC1918] will be filtered out. This prevents exposure of internal network details, at the cost of requiring relay usage even for intranet calls, if the NAT does not allow hairpinning as described in [RFC4787], section 6.

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 its relay servers, this could obfuscate location to a metro or possibly even global level.

Although it can be overridden by local policy, the default ICE candidate policy MUST be set to allow all candidates, as this minimizes use of application STUN/TURN server resources.

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]. By specifying a policy from the list below, the application can control how aggressively it will try to BUNDLE media streams together. The set of available policies is as follows:

balanced: The application will BUNDLE all media streams of the same type together. That is, if there are multiple audio and multiple video MediaStreamTracks attached to a PeerConnection, all but the first audio and video tracks will be marked as bundle-only, and candidates will only be gathered for N media streams, where N is the number of distinct media types. When talking to a non-BUNDLE-aware endpoint, only the non-bundle-only streams will be negotiated. This policy balances desire to multiplex with the need to ensure basic audio and video still works in legacy cases. Data channels will be in a separate bundle group.

max-compat: The application will offer BUNDLE, but mark none of its streams 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: The application will BUNDLE all of its media streams, including data channels, on a single transport. 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.

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 browser 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 browser will only gather RTP candidates. [[OPEN ISSUE: how should the answerer behave. https://github.com/rtcweb-wg/jsep/issues/114]] This halves the number of candidates that the offerer needs to gather.

4.1.2. 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 local MediaStreams attached 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 should allow for the following manipulations to be performed:

- To indicate support for a media type even if no MediaStreamTracks of that type have been added to the session (e.g., an audio call that wants to receive video.)

- 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 removing MediaStreams, 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. Because this method may need to inspect the system state to determine the currently available resources, it may be implemented as an async operation.

Calling this method may do things such as generate new ICE credentials, but does not result in candidate gathering, or cause media to start or stop flowing.

4.1.3. 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 local MediaStreams attached 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. Because this method may need to inspect the system state to determine the

currently available resources, it may need to be implemented as an async operation.

Calling this method may do things such as generate new ICE credentials, but does not trigger candidate gathering or change media state.

4.1.4. SessionDescriptionType

Session description objects (RTCSessionDescription) may be of type "offer", "pranswer", or "answer". 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 a "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 state, as described in Section 4.1.4.2. The contents MUST be empty.

4.1.4.1. Use of Provisional Answers

Most web 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 web application would be to create and send an "inactive" final answer immediately after receiving the offer. Later, when the called user actually accepts the call, the application can create a new "sendrecv" offer to update the previous offer/answer pair and start the media flow. While this could also be done with an inactive "pranswer", followed by a sendrecv "answer", the initial "pranswer" leaves the offer-answer exchange open, which means that neither side can send an updated offer during this time.

As an example, consider a typical web application that will set up a data channel, an audio channel, and a video channel. When an endpoint receives an offer with these channels, it could send an answer accepting the data channel for two-way data, and accepting the audio and video tracks as inactive or receive-only. It could then ask the user to accept the call, acquire the local media streams, and send a new offer to the remote side moving the audio and video to be two-way media. By the time the human has accepted the call and triggered the new offer, it is likely that the ICE and DTLS handshaking for all the channels will already have finished.

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.

4.1.4.2. 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 modified local and/or remote description back to their previous values. 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 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).

4.1.5. setLocalDescription

The setLocalDescription method instructs the PeerConnection to apply the supplied SDP blob as its local configuration. The type field indicates whether the blob 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 old and new local descriptions (e.g. support codecs that exist in both descriptions) until a final answer is received, at which point the PeerConnection can fully adopt the new local description, or roll back to the old description if the remote side denied the change.

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 them.

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

4.1.6. setRemoteDescription

The setRemoteDescription method instructs the PeerConnection to apply the supplied SDP blob as the desired remote configuration. As in

setLocalDescription, the type field of the indicates how the blob 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 are available to send, this will result in the starting of media transmission.

4.1.7. localDescription

The localDescription method returns a copy of the current local configuration, i.e. what was most recently passed to setLocalDescription, plus any local candidates that have been generated by the ICE Agent.

[[OPEN ISSUE: Do we need to expose accessors for both the current and proposed local description? https://github.com/rtcweb-wg/jsep/issues/16]]

A null object will be returned if the local description has not yet been established.

4.1.8. remoteDescription

The remoteDescription method returns a copy of the current remote configuration, i.e. what was most recently passed to setRemoteDescription, plus any remote candidates that have been supplied via processIceMessage.

[[OPEN ISSUE: Do we need to expose accessors for both the current and proposed remote description? https://github.com/rtcweb-wg/jsep/issues/16]]

A null object will be returned if the remote description has not yet been established.

4.1.9. 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 3.4.2, 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.

4.1.10. 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:

- Any changes to the STUN/TURN servers to use affect the next gathering phase. If gathering has already occurred, 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, any existing candidates will be discarded, and new candidates will be gathered from the new servers.

- Any changes to the ICE candidate policy also affect the next gathering phase, in similar fashion to the server changes described above. Note though that 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.

- 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.

- 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, and may result in a change to media state if it results in connectivity being established.

4.1.11. addIceCandidate

The addIceCandidate method provides a remote candidate to the ICE Agent, which, if parsed successfully, will be added to the remote description according to the rules defined for Trickle ICE. Connectivity checks will be sent to the new candidate.

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.

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.

The first set of specifications is the "mandatory-to-implement" set. All implementations must support these behaviors, but may not use all of them if the remote side, which may not be a JSEP endpoint, does not support them.

The second set of specifications is the "mandatory-to-use" set. The local JSEP endpoint and any remote endpoint must indicate support for these specifications in their session descriptions.

5.1.1. Implementation Requirements

This list of mandatory-to-implement specifications is derived from the requirements outlined in [I-D.ietf-rtcweb-rtp-usage].

R-1 [RFC4566] is the base SDP specification and MUST be implemented.

R-2 [RFC5764] MUST be supported for signaling the UDP/TLS/RTP/SAVPF [RFC5764] and TCP/DTLS/RTP/SAVPF [I-D.nandakumar-mmusic-proto-iana-registration] RTP profiles.

R-3 [RFC5245] MUST be implemented for signaling the ICE credentials and candidate lines corresponding to each media stream. The ICE implementation MUST be a Full implementation, not a Lite implementation.

R-4 [RFC5763] MUST be implemented to signal DTLS certificate fingerprints.

R-5 [RFC4568] MUST NOT be implemented to signal SDES SRTP keying information.

R-6 The [RFC5888] grouping framework MUST be implemented for signaling grouping information, and MUST be used to identify m= lines via the a=mid attribute.

R-7 [I-D.ietf-mmusic-msid] MUST be supported, in order to signal associations between RTP objects and W3C MediaStreams and MediaStreamTracks in a standard way.

R-8 The bundle mechanism in [I-D.ietf-mmusic-sdp-bundle-negotiation] MUST be supported to signal the ability to multiplex RTP streams on a single UDP port, in order to avoid excessive use of port number resources.

R-9 The SDP attributes of "sendonly", "recvonly", "inactive", and "sendrecv" from [RFC4566] MUST be implemented to signal information about media direction.

R-10 [RFC5576] MUST be implemented to signal RTP SSRC values and grouping semantics.

R-11 [RFC4585] MUST be implemented to signal RTCP based feedback.

R-12 [RFC5761] MUST be implemented to signal multiplexing of RTP and RTCP.

R-13 [RFC5506] MUST be implemented to signal reduced-size RTCP messages.

R-14 [RFC4588] MUST be implemented to signal RTX payload type associations.

R-15 [RFC3556] with bandwidth modifiers MAY be supported for specifying RTCP bandwidth as a fraction of the media bandwidth, RTCP fraction allocated to the senders and setting maximum media bit-rate boundaries.

R-16 TODO: any others?

As required by [RFC4566], Section 5.13, JSEP implementations MUST ignore unknown attribute (a=) lines.

5.1.2. Usage Requirements

All session descriptions handled by JSEP endpoints, 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.

R-1 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.

R-2 DTLS [RFC6347] or DTLS-SRTP [RFC5763], MUST be used, as appropriate for the media type, as specified in [I-D.ietf-rtcweb-security-arch]

5.1.3. Profile Names and Interoperability

For media m= sections, JSEP endpoints MUST support both the "UDP/TLS/RTP/SAVPF" and "TCP/DTLS/RTP/SAVPF" profiles and MUST indicate one of these two profiles for each media m= line they produce in an offer. For data m= sections, JSEP endpoints must support both the "UDP/DTLS/SCTP" and "TCP/DTLS/SCTP" profiles and MUST indicate one of these two profiles for each data m= line they produce in an offer. Because ICE can select either TCP or UDP transport depending on network conditions, both advertisements are 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 endpoints 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 the "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 AVP timing, i.e., "trr-int=4".

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

Note that re-offers by JSEP endpoints 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 "-". The value of the <sess-id> field SHOULD be a cryptographically random number. To ensure uniqueness, this number SHOULD be at least 64 bits long. The value of the <sess-version> field SHOULD be zero. 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="), Bandwidth ("b="), 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=msid-semantic:WMS" line MUST be added, as specified in [I-D.ietf-mmusic-msid], Section 4.

The next step is to generate m= sections, as specified in [RFC4566] Section 5.14, for each MediaStreamTrack that has been added to the PeerConnection via the addStream method. (Note that this method takes a MediaStream, which can contain multiple MediaStreamTracks, and therefore multiple m= sections can be generated even if addStream is only called once.) m=sections MUST be sorted first by the order in which the MediaStreams were added to the PeerConnection, and then by the alphabetical ordering of the media type for the MediaStreamTrack. For example, if a MediaStream containing both an audio and a video MediaStreamTrack is added to a PeerConnection, the resultant m=audio section will precede the m=video section. If a second MediaStream containing an audio MediaStreamTrack was added, it would follow the m=video section.

Each m= section, provided it is not being bundled into another m= section, MUST generate a unique set of ICE credentials and gather its own unique set of ICE candidates. Otherwise, it MUST use the same ICE credentials and candidates as the m= section into which it is being bundled. Note that this means that for offers, any m= sections which are not bundle-only MUST have unique ICE credentials and candidates, since it is possible that the answerer will accept them without bundling them.

For DTLS, all m= sections MUST use the certificate for the identity that has been specified for the PeerConnection; as a result, they MUST all have the same [RFC4572] fingerprint value, or this value MUST be a session-level attribute.

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

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

- To properly indicate use of DTLS, the <proto> field MUST be set to "UDP/TLS/RTP/SAVPF", as specified in [RFC5764], Section 8, if the default candidate uses UDP transport, or "TCP/DTLS/RTP/SAVPF", as specified in[I-D.nandakumar-mmusic-proto-iana-registration] if the default candidate uses TCP transport.

The m= line MUST be followed immediately by a "c=" line, as specified in [RFC4566], Section 5.7. Again, as no candidates have yet been gathered, the "c=" line must contain the "dummy" value "IN IP6 ::", as defined in [I-D.ietf-mmusic-trickle-ice], Section 5.1.

Each m= section MUST include the following attribute lines:

- An "a=mid" line, as specified in [RFC5888], Section 4. When generating mid values, it is RECOMMENDED that the values 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 11.

- An "a=rtcp" line, as specified in [RFC3605], Section 2.1, containing the dummy value "9 IN IP6 ::", because no candidates have yet been gathered.

- An "a=msid" line, as specified in [I-D.ietf-mmusic-msid], Section 2.

- An "a=sendrecv" line, as specified in [RFC3264], Section 5.1.

- For each supported codec, "a=rtpmap" and "a=fmtp" lines, as specified in [RFC4566], Section 6. The audio and video codecs that MUST be supported are specified in [I-D.ietf-rtcweb-audio] (see Section 3) and [I-D.ietf-rtcweb-video] (see Section 5).

- If this m= section is for media with configurable frame sizes, e.g. audio, an "a=maxptime" line, indicating the smallest of the maximum supported frame sizes out of all codecs included above, as specified in [RFC4566], Section 6.

- 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.

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

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

- An "a=fingerprint" line, as specified in [RFC4572], Section 5; the algorithm used for the fingerprint MUST match that used in the certificate signature.

- 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".

- An "a=rtcp-mux" line, as specified in [RFC5761], Section 5.1.1.

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

- 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.

- An "a=ssrc" line, as specified in [RFC5576], Section 4.1, indicating the SSRC to be used for sending media, along with the mandatory "cname" source attribute, as specified in Section 6.1, indicating the CNAME for the source. The CNAME must be generated in accordance with [RFC7022]. [OPEN ISSUE: How are CNAMEs specified for MSTs? Are they randomly generated for each

MediaStream? If so, can two MediaStreams be synced? See: https://github.com/rtcweb-wg/jsep/issues/4]

- If RTX is supported for this media type, another "a=ssrc" line with the RTX SSRC, and an "a=ssrc-group" line, as specified in [RFC5576], section 4.2, with semantics set to "FID" and including the primary and RTX SSRCs.

- If FEC is supported for this media type, another "a=ssrc" line with the FEC SSRC, and an "a=ssrc-group" line with semantics set to "FEC-FR" and including the primary and FEC SSRCs, as specified in [RFC5956], section 4.3. For simplicity, if both RTX and FEC are supported, the FEC SSRC MUST be the same as the RTX SSRC.

- [OPEN ISSUE: Handling of a=imageattr]

- 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.

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" if the default candidate uses UDP transport, or "TCP/DTLS/SCTP" if the default candidate uses TCP transport [I-D.ietf-mmusic-sctp-sdp]. The "fmt" value MUST be set to the SCTP port number, as specified in Section 4.1. [TODO: update this to use a=sctp-port, as indicated in the latest data channel docs]

Within the data m= section, the "a=mid", "a=ice-ufrag", "a=ice-passwd", "a=ice-options", "a=candidate", "a=fingerprint", and "a=setup" lines MUST be included as mentioned above, along with an "a=sctpmap" line referencing the SCTP port number and specifying the application protocol indicated in [I-D.ietf-rtcweb-data-protocol]. [OPEN ISSUE: the -01 of this document is missing this information.]

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 browser offers all m= sections as one BUNDLE group. However, whether the m= sections are bundle-only or not depends on the BUNDLE policy.

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:

- "a=crypto"
- "a=key-mgmt"
- "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:

- 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, including the addition of ICE candidates.

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:

- The "s=" and "t=" lines MUST stay the same.
- 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. Each "a=rtcp" attribute line MUST also be filled in with the port and address of the

appropriate default candidate, either the default RTP or RTCP candidate, depending on whether RTCP multiplexing is currently active or not. Note that if RTCP multiplexing is being offered, but not yet active, the default RTCP candidate MUST be used, as indicated in [RFC5761], section 5.1.3. In each case, if no candidates of the desired type have yet been gathered, dummy values MUST be used, as described above.

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

- 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 5.2.3.3 was specified.

- Within each m= section, for each candidate that has been gathered during the most recent gathering phase (see Section 3.4.1), an "a=candidate" line MUST be added, as specified 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-mmusic-trickle-ice], Section 9.3.

- For MediaStreamTracks that are still present, the "a=msid", "a=ssrc", and "a=ssrc-group" lines MUST stay the same.

- If any MediaStreamTracks have been removed, either through the removeStream method or by removing them from an added MediaStream, their m= sections MUST be marked as recvonly by changing the value of the [RFC3264] directional attribute to "a=recvonly". The "a=msid", "a=ssrc", and "a=ssrc-group" lines MUST be removed from the associated m= sections.

- If any MediaStreamTracks have been added, and there exist m= sections of the appropriate media type with no associated MediaStreamTracks (i.e. as described in the preceding paragraph), those m= sections MUST be recycled by adding the new MediaStreamTrack to the m= section. This is done by adding the necessary "a=msid", "a=ssrc", and "a=ssrc-group" lines to the recycled m= section, and removing the "a=recvonly" attribute.

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, along with these exceptions: [OPEN ISSUE: should this be permitted in the remote-pranswer state?]

- If a m= section exists in the current local description, but does not have an associated local MediaStreamTrack (possibly because said MediaStreamTrack was removed since the last exchange), a m= section MUST still be generated in the new offer, as indicated in [RFC3264], Section 8. The disposition of this section will depend on the state of the remote MediaStreamTrack associated with this m= section. If one exists, and it is still in the "live" state, the new m= section MUST be marked as "a=recvonly", with no "a=msid" or related attributes present. If no remote MediaStreamTrack exists, or it is in the "ended" state, the m= section MUST be marked as rejected, by setting the port to zero, as indicated in [RFC3264], Section 8.2.

- If any MediaStreamTracks have been added, and there exist recvonly m= sections of the appropriate media type with no associated MediaStreamTracks, or rejected m= sections of any media type, those m= sections MUST be recycled, and a local MediaStreamTrack associated with these recycled m= sections until all such existing m= sections have been used. This includes any recvonly or rejected m= sections created by the preceding paragraph.

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:

- The m= line and corresponding "a=rtpmap" and "a=fmtp" lines MUST only include codecs present in the remote description.

- The RTP header extensions MUST only include those that are present in the remote description.

- The RTCP feedback extensions MUST only include those that are present in the remote description.

- The "a=rtcp-mux" line MUST only be added if present in the remote description.

- The "a=rtcp-rsize" line MUST only be added if present in the remote description.

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.

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. OfferToReceiveAudio

If the "OfferToReceiveAudio" option is specified, with an integer value of N, and M audio MediaStreamTracks have been added to the PeerConnection, the offer MUST include N non-rejected m= sections with media type "audio", even if N is greater than M. This allows the offerer to receive audio, including multiple independent streams, even when not sending it; accordingly, the directional attribute on the N-M audio m= sections without associated MediaStreamTracks MUST be set to recvonly.

If N is set to a value less than M, the offer MUST mark the m= sections associated with the M-N most recently added (since the last setLocalDescription) MediaStreamTracks as sendonly. This allows the offerer to indicate that it does not want to receive audio on some or all of its newly created streams. For m= sections that have previously been negotiated, this setting has no effect. [TODO: refer to RTCRtpSender in the future]

For backwards compatibility with pre-standard versions of this specification, a value of "true" is interpreted as equivalent to N=1, and "false" as N=0.

5.2.3.2. OfferToReceiveVideo

If the "OfferToReceiveVideo" option is specified, with an integer value of N, and M video MediaStreamTracks have been added to the PeerConnection, the offer MUST include N non-rejected m= sections with media type "video", even if N is greater than M. This allows the offerer to receive video, including multiple independent streams, even when not sending it; accordingly, the directional attribute on the N-M video m= sections without associated MediaStreamTracks MUST be set to recvonly.

If N is set to a value less than M, the offer MUST mark the m= sections associated with the M-N most recently added (since the last setLocalDescription) MediaStreamTracks as sendonly. This allows the offerer to indicate that it does not want to receive video on some or all of its newly created streams. For m= sections that have previously been negotiated, this setting has no effect. [TODO: refer to RTCRtpSender in the future]

For backwards compatibility with pre-standard versions of this specification, a value of "true" is interpreted as equivalent to N=1, and "false" as N=0.

5.2.3.3. 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.

5.2.3.4. 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, the "usedtx=1" parameter would be specified 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.

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.

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 SHALL be considered to be present in each m= section.

The next step is to go through each offered m= section. If there is a local MediaStreamTrack of the same type which has been added to the PeerConnection via addStream and not yet associated with a m= section, and the specific m= section is either sendrecv or recvonly, the MediaStreamTrack will be associated with the m= section at this time. MediaStreamTracks are assigned to m= sections using the canonical order described in Section 5.2.1. If there are more m= sections of a certain type than MediaStreamTracks, some m= sections will not have an associated MediaStreamTrack. If there are more MediaStreamTracks of a certain type than compatible m= sections, only the first N MediaStreamTracks will be able to be associated in the constructed answer. The remainder will need to be associated in a subsequent offer.

For each offered m= section, if the associated remote MediaStreamTrack has been stopped, and is therefore in state "ended", and no local MediaStreamTrack has been associated, 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.

Provided that is not the case, 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 have yet been gathered, the "dummy" port value of 9 (Discard) MUST be used, as indicated in [I-D.ietf-mmusic-trickle-ice], Section 5.1.

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

The m= line MUST be followed immediately by a "c=" line, as specified in [RFC4566], Section 5.7. Again, as no candidates have yet been gathered, the "c=" line must contain the "dummy" value "IN IP6 ::", as defined in [I-D.ietf-mmusic-trickle-ice], 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 include the following:

- 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.

- An "a=rtcp" line, as specified in [RFC3605], Section 2.1, containing the dummy value "9 IN IP6 ::", because no candidates have yet been gathered.

- If a local MediaStreamTrack has been associated, an "a=msid" line, as specified in [I-D.ietf-mmusic-msid], Section 2.

- Depending on the directionality of the offer, the disposition of any associated remote MediaStreamTrack, and the presence of an associated local MediaStreamTrack, the appropriate directionality attribute, as specified in [RFC3264], Section 6.1. If the offer was sendrecv, and the remote MediaStreamTrack is still "live", and there is a local MediaStreamTrack that has been associated, the directionality MUST be set as sendrecv. If the offer was sendonly, and the remote MediaStreamTrack is still "live", the directionality MUST be set as recvonly. If the offer was recvonly, and a local MediaStreamTrack has been associated, the directionality MUST be set as sendonly. If the offer was inactive, the directionality MUST be set as inactive.

- For each supported codec that is present in the offer, "a=rtpmap" and "a=fmtp" lines, as specified in [RFC4566], Section 6, and [RFC3264], Section 6.1. The audio and video codecs that MUST be supported are specified in [I-D.ietf-rtcweb-audio] (see Section 3) and [I-D.ietf-rtcweb-video] (see Section 5). Note that for simplicity, the answerer MAY use different payload types for codecs than the offerer, as it is not prohibited by Section 6.1.

- If this m= section is for media with configurable frame sizes, e.g. audio, an "a=maxptime" line, indicating the smallest of the

maximum supported frame sizes out of all codecs included above, as specified in [RFC4566], Section 6.

- 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.

- 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.

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

- If the "trickle" ICE option is present in the offer, an "a=ice-options" line, with the "trickle" option, as specified in [I-D.ietf-mmusic-trickle-ice], Section 4.

- An "a=fingerprint" line, as specified in [RFC4572], Section 5; the algorithm used for the fingerprint MUST match that used in the certificate signature.

- 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.

- If present in the offer, an "a=rtcp-mux" line, as specified in [RFC5761], Section 5.1.1. If the "require" RTCP multiplexing policy is set and no "a=rtcp-mux" line is present in the offer, then the m=line MUST be marked as rejected by setting the port in the m= line to zero, as indicated in [RFC3264], Section 6.

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

- 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.

- 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.

- If a local MediaStreamTrack has been associated, an "a=ssrc" line, as specified in [RFC5576], Section 4.1, indicating the SSRC to be used for sending media.

- If a local MediaStreamTrack has been associated, and RTX has been negotiated for this m= section, another "a=ssrc" line with the RTX SSRC, and an "a=ssrc-group" line, as specified in [RFC5576], section 4.2, with semantics set to "FID" and including the primary and RTX SSRCs.

- If a local MediaStreamTrack has been associated, and FEC has been negotiated for this m= section, another "a=ssrc" line with the FEC SSRC, and an "a=ssrc-group" line with semantics set to "FEC-FR" and including the primary and FEC SSRCs, as specified in [RFC5956], section 4.3. For simplicity, if both RTX and FEC are supported, the FEC SSRC MUST be the same as the RTX SSRC.

- [OPEN ISSUE: Handling of a=imageattr]

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> field MUST be set to exactly match the field in the offer; the "fmt" value MUST be set to the SCTP port number, as specified in Section 4.1. [TODO: update this to use a=sctp-port, as indicated in the latest data channel docs]

Within the data m= section, the "a=mid", "a=ice-ufrag", "a=ice-passwd", "a=ice-options", "a=candidate", "a=fingerprint", and "a=setup" lines MUST be included as mentioned above, along with an "a=sctpmap" line referencing the SCTP port number and specifying the application protocol indicated in [I-D.ietf-rtcweb-data-protocol]. [OPEN ISSUE: the -01 of this document is missing this information.]

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:

- 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:

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

- 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. Each "a=rtcp" attribute line MUST also be filled in with the port and address of the appropriate default candidate, either the default RTP or RTCP candidate, depending on whether RTCP multiplexing is enabled in the answer. In each case, if no candidates of the desired type have yet been gathered, dummy values MUST be used, as described in the initial answer section above.

- Each "a=ice-ufrag" and "a=ice-pwd" line MUST stay the same.

- Within each m= section, for each candidate that has been gathered during the most recent gathering phase (see Section 3.4.1), an "a=candidate" line MUST be added, as specified 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-mmusic-trickle-ice], Section 9.3.

- For MediaStreamTracks that are still present, the "a=msid", "a=ssrc", and "a=ssrc-group" lines MUST stay the same.

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 OfferToReceiveAudio, OfferToReceiveVideo, and IceRestart options mentioned in Section 5.2.3 are meaningless in the context of generating an answer, as there is no need to generate extra m= lines in an answer, and ICE credentials will automatically be changed for all m= lines 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 5.2.3.4.

5.4. 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.
- Next, the SessionDescription is parsed into a data structure, as described in the Section 5.6 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 5.7 section below.

5.5. 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.
- Next, the SessionDescription is parsed into a data structure, as described in the Section 5.6 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. Parsing a Session Description

[The behavior described herein is a draft version, and needs more discussion to resolve various open issues.]

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 its defining ABNF [TODO: reference], check that it conforms to [RFC4566] and [RFC3264] semantics, and then either parse and store or discard the provided value, as described below. [TODO: ensure that every line is listed below.] If the line is not well-formed, or cannot be parsed as described, the parser MUST stop with an error and reject the session description. This ensures that implementations do not accidentally misinterpret ambiguous SDP.

5.6.1. 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.

For non-attribute (non-"a=") lines, their sequencing, syntax, and semantics, are checked, as mentioned above. The following lines are not meaningful in the JSEP context and MAY be discarded once they have been checked.

TODO

The remaining lines are processed as follows:

The "c=" line MUST be parsed and stored.

[OPEN ISSUE: For example, because session-level bandwidth is ambiguous when multiple media streams are present, a "b=" line at session level is not useful and its value SHOULD be ignored. [OPEN ISSUE: is this WG consensus? Are there other non-a= lines that we need to do more than just syntactical validation, e.g. v=?]

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.

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

- TODO: msid-semantic, identity, rtcp-rsize, rtcp-mux, and any other attribs valid at session level.

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

5.6.2. Media Section Parsing

Like the session-level lines, the media session 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:

- The "c=" line, if present, MUST be parsed as specified in [RFC4566], Section 5.7, and its contents stored.

- 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:

- 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 the "m=" proto value indicates use of RTP, as decribed in the Section 5.1.3 section above, the following attribute lines MUST be processed:

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

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

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

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

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

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

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

- If present, a single "a=rtcp-mux" line MUST be parsed as specified in [RFC5761], Section 5.1.1, and its presence or absence flagged and stored.

- TODO: a=rtcp-rsize, a=rtcp, a=msid, a=candidate, a=end-of-candidates

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

- 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.

- 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.

- TODO: max message size

5.6.3. 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:

- For each m= section, valid values for each of the mandatory-to-use features enumerated in Section 5.1.2 MUST be present. These values MAY either be present at the media level, or inherited from the session level.

  - ICE ufrag and password values

  - DTLS fingerprint and setup values

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

- The session description must follow the rules defined in [RFC3264], Section 6.

- For each m= section, the protocol value MUST exactly match the protocol value in the corresponding m= section in the associated offer.

5.7. Applying a Local Description

The following steps are performed at the media engine level to apply a local description.

First, the parsed parameters are checked to ensure that any modifications performed fall within those explicitly permitted by Section 6; otherwise, processing MUST stop and an error MUST be returned.

Next, media sections are processed. For each media 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.

- TODO

Finally, if this description is of type "pranswer" or "answer", follow the processing defined in the Section 5.9 section below.

5.8. Applying a Remote Description

TODO

5.9. Applying an Answer

TODO

6. Configurable SDP Parameters

It is possible to change elements in the SDP returned from createOffer before passing it to setLocalDescription. When an implementation receives modified SDP it MUST either:

- Accept the changes and adjust its behavior to match the SDP.
- Reject the changes and return an error via the error callback.

Changes MUST NOT be silently ignored.

The following elements of the SDP media description MUST NOT be changed between the createOffer and the setLocalDescription (or between the createAnswer and the setLocalDescription), since they reflect transport attributes that are solely under browser control, and the browser MUST NOT honor an attempt to change them:

- The number, type and port number of m= lines.
- The generated ICE credentials (a=ice-ufrag and a=ice-pwd).
- The set of ICE candidates and their parameters (a=candidate).
- The DTLS fingerprint(s) (a=fingerprint).

The following modifications, if done by the browser to a description between createOffer/createAnswer and the setLocalDescription, MUST be honored by the browser:

- Remove or reorder codecs (m=)

The following parameters may be controlled by options passed into createOffer/createAnswer. As an open issue, these changes may also be be performed by manipulating the SDP returned from createOffer/createAnswer, as indicated above, as long as the capabilities of the endpoint are not exceeded (e.g. asking for a resolution greater than what the endpoint can encode):

- [[OPEN ISSUE: This is a placeholder for other modifications, which we may continue adding as use cases appear.]]

Implementations MAY choose to either honor or reject any elements not listed in the above two categories, but must do so explicitly as described at the beginning of this section. Note that future standards may add new SDP elements to the list of elements which must be accepted or rejected, but due to version skew, applications must be prepared for implementations to accept changes which must be rejected and vice versa.

The application can also modify the SDP to reduce the capabilities in the offer it sends to the far side or the offer that it installs from the far side in any way the application sees fit, as long as it is 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 actually in the offer.

As always, the application is solely responsible for what it sends to the other party, and all incoming SDP will be processed by the browser 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.

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 browsers and does not use trickle ICE. The example in the following section provides a more realistic example of what would happen in a normal browser to browser connection.

The flow shows Alice's browser initiating the session to Bob's browser. 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 policy of balanced.

```
//                  set up local media state
AliceJS->AliceUA:   create new PeerConnection
AliceJS->AliceUA:   addStream with stream containing 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 value of localDescription

//                  |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:       onaddstream event with remoteStream

//                  Bob accepts call
BobJS->BobUA:       addStream with local media
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 value of localDescription

//                  |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:   onaddstream event with remoteStream

//                  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=msid-semantic:WMS
   a=group:BUNDLE a1 v1
   m=audio 56500 UDP/TLS/RTP/SAVPF 96 0 8 97 98
   c=IN IP4 192.0.2.1
   a=mid:a1
   a=rtcp:56501 IN IP4 192.0.2.1
   a=msid:47017fee-b6c1-4162-929c-a25110252400
          f83006c5-a0ff-4e0a-9ed9-d3e6747be7d9
   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=ice-ufrag:ETEn1v9DoTMB9J4r
   a=ice-pwd:OtSK0WpNtpUjkY4+86js7ZQl
   a=ice-options:trickle
   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=rtcp-mux
   a=rtcp-rsize
   a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level
   a=extmap:2 urn:ietf:params:rtp-hdrext:sdes:mid
   a=ssrc:1732846380 cname:EocUG1f0fcg/yvY7
   a=candidate:3348148302 1 udp 2113937151 192.0.2.1 56500
               typ host
   a=candidate:3348148302 2 udp 2113937151 192.0.2.1 56501
               typ host
   a=end-of-candidates

   m=video 56502 UDP/TLS/RTP/SAVPF 100 101
   c=IN IP4 192.0.2.1
   a=rtcp:56503 IN IP4 192.0.2.1
   a=mid:v1
   a=msid:61317484-2ed4-49d7-9eb7-1414322a7aae
          f30bdb4a-5db8-49b5-bcdc-e0c9a23172e0
   a=sendrecv
   a=rtpmap:100 VP8/90000
   a=rtpmap:101 rtx/90000
   a=fmtp:101 apt=100
   a=ice-ufrag:BGKkWnG5GmiUpdIV
   a=ice-pwd:mqyWsAjvtKwTGnvhPztQ9mIf
   a=ice-options:trickle
   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=rtcp-mux
   a=rtcp-rsize
   a=extmap:3 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=ssrc:1366781083 cname:EocUG1f0fcg/yvY7
   a=ssrc:1366781084 cname:EocUG1f0fcg/yvY7
   a=ssrc-group:FID 1366781083 1366781084
   a=candidate:3348148302 1 udp 2113937151 192.0.2.1 56502
               typ host
   a=candidate:3348148302 2 udp 2113937151 192.0.2.1 56503
               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=msid-semantic:WMS
   m=audio 20000 UDP/TLS/RTP/SAVPF 96 0 8 97 98
   c=IN IP4 192.0.2.2
   a=mid:a1
   a=rtcp:20000 IN IP4 192.0.2.2
   a=msid:PI39StLS8W7ZbQl1sJsWUXkr3Zf12fJUvzQ1
          PI39StLS8W7ZbQl1sJsWUXkr3Zf12fJUvzQ1a0
   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=ice-ufrag:6sFvz2gdLkEwjZEr
   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=rtcp-mux
   a=rtcp-rsize
   a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level
   a=ssrc:3429951804 cname:Q/NWs1ao1HmN4Xa5
   a=candidate:2299743422 1 udp 2113937151 192.0.2.2 20000
               typ host
```

```
a=end-of-candidates

m=video 20001 UDP/TLS/RTP/SAVPF 100 101
c=IN IP4 192.0.2.2
a=rtcp 20001 IN IP4 192.0.2.2
a=mid:v1
a=msid:PI39StLS8W7ZbQl1sJsWUXkr3Zf12fJUvzQ1
       PI39StLS8W7ZbQl1sJsWUXkr3Zf12fJUvzQ1v0
a=sendrecv
a=rtpmap:100 VP8/90000
a=rtpmap:101 rtx/90000
a=fmtp:101 apt=100
a=ice-ufrag:6sFvz2gdLkEwjZEr
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=rtcp-mux
a=rtcp-rsize
a=rtcp-fb:100 ccm fir
a=rtcp-fb:100 nack
a=rtcp-fb:100 nack pli
a=ssrc:3229706345 cname:Q/NWs1ao1HmN4Xa5
a=ssrc:3229706346 cname:Q/NWs1ao1HmN4Xa5
a=ssrc-group:FID 3229706345 3229706346
a=candidate:2299743422 1 udp 2113937151 192.0.2.2 20001
            typ host
a=end-of-candidates
```

7.2. Normal Examples

This section shows a typical example of a session between two browsers setting up an audio channel and a data channel. 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. Later, two video flows, one for the presenter and one for screen sharing, are added to the session. This example shows Alice's browser initiating the session to Bob's browser. The messages from Alice's JS to Bob's JS are assumed to flow over some signaling protocol via a web server.

```
//                 set up local media state
AliceJS->AliceUA:  create new PeerConnection
AliceJS->AliceUA:  addStream that contains 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:       onaddstream with audio track from Alice

//                  candidates are sent to Bob
AliceUA->AliceJS:   onicecandidate event with |candidate-B1| (host)
AliceJS->WebServer: signaling with |candidate-B1|
AliceUA->AliceJS:   onicecandidate event with |candidate-B2| (srflx)
AliceJS->WebServer: signaling with |candidate-B2|
AliceUA->AliceJS:   onicecandidate event with |candidate-B3| (relay)
AliceJS->WebServer: signaling with |candidate-B3|

WebServer->BobJS:   signaling with |candidate-B1|
BobJS->BobUA:       addIceCandidate with |candidate-B1|
WebServer->BobJS:   signaling with |candidate-B2|
BobJS->BobUA:       addIceCandidate with |candidate-B2|
WebServer->BobJS:   signaling with |candidate-B3|
BobJS->BobUA:       addIceCandidate with |candidate-B3|

//                  Bob accepts call
BobJS->BobUA:       addStream with local audio stream
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:   onaddstream event with audio track from Bob

//                  candidates are sent to Alice
BobUA->BobJS:       onicecandidate event with |candidate-B4| (host)
BobJS->WebServer:   signaling with |candidate-B4|
BobUA->BobJS:       onicecandidate event with |candidate-B5| (srflx)
BobJS->WebServer:   signaling with |candidate-B5|
BobUA->BobJS:       onicecandidate event with |candidate-B6| (relay)
BobJS->WebServer:   signaling with |candidate-B6|

WebServer->AliceJS: signaling with |candidate-B4|
AliceJS->AliceUA:   addIceCandidate with |candidate-B4|
WebServer->AliceJS: signaling with |candidate-B5|
AliceJS->AliceUA:   addIceCandidate with |candidate-B5|
WebServer->AliceJS: signaling with |candidate-B6|
```

```
  AliceJS->AliceUA:   addIceCandidate with |candidate-B6|

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

  //                  media is flowing between browsers
  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:       addStream with first video stream
  BobJS->BobUA:       addStream 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:   onaddstream event with first video stream
  AliceUA->AliceJS:   onaddstream event with second video stream
  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 browsers
  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=msid-semantic:WMS
   a=group:BUNDLE a1 d1
   m=audio 9 UDP/TLS/RTP/SAVPF 96 0 8 97 98
   c=IN IP6 ::
   a=rtcp:9 IN IP6 ::
   a=mid:a1
   a=msid:57017fee-b6c1-4162-929c-a25110252400
          e83006c5-a0ff-4e0a-9ed9-d3e6747be7d9
   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=ice-ufrag:ATEn1v9DoTMB9J4r
   a=ice-pwd:AtSK0WpNtpUjkY4+86js7ZQl
   a=ice-options:trickle
   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=rtcp-mux
   a=rtcp-rsize
   a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level
   a=extmap:2 urn:ietf:params:rtp-hdrext:sdes:mid
   a=ssrc:1732846380 cname:FocUG1f0fcg/yvY7

   m=application 9 UDP/DTLS/SCTP webrtc-datachannel
   c=IN IP6 ::
   a=mid:d1
   a=fmtp:webrtc-datachannel max-message-size=65536
   a=sctp-port 5000
   a=ice-ufrag:ATEn1v9DoTMB9J4r
   a=ice-pwd:AtSK0WpNtpUjkY4+86js7ZQl
   a=ice-options:trickle
   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
```

The SDP for |candidate-B1| looks like:

```
   candidate:109270923 1 udp 2122194687 192.168.1.2 51556 typ host
```

The SDP for |candidate-B2| looks like:

```
candidate:4036177503 1 udp 1685987071 11.22.33.44 52546 typ srflx
          raddr 192.168.1.2 rport 51556
```

The SDP for |candidate-B3| looks like:

```
candidate:3671762466 1 udp 41819903 22.33.44.55 61405 typ relay
          raddr 11.22.33.44 rport 52546
```

The SDP for |answer-B1| looks like:

```
   v=0
   o=- 7729291447651054566 1 IN IP4 0.0.0.0
   s=-
   t=0 0
   a=msid-semantic:WMS
   a=group:BUNDLE a1 d1
   m=audio 9 UDP/TLS/RTP/SAVPF 96 0 8 97 98
   c=IN IP6 ::
   a=rtcp:9 IN IP6 ::
   a=mid:a1
   a=msid:QI39StLS8W7ZbQl1sJsWUXkr3Zf12fJUvzQ1
          QI39StLS8W7ZbQl1sJsWUXkr3Zf12fJUvzQ1a0
   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=ice-ufrag:7sFvz2gdLkEwjZEr
   a=ice-pwd:dOTZKZNVlO9RSGsEGM63JXT2
   a=ice-options:trickle
   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=rtcp-mux
   a=rtcp-rsize
   a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level
   a=extmap:2 urn:ietf:params:rtp-hdrext:sdes:mid
   a=ssrc:4429951804 cname:Q/NWs1ao1HmN4Xa5

   m=application 9 UDP/DTLS/SCTP webrtc-datachannel
   c=IN IP6 ::
   a=mid:d1
   a=fmtp:webrtc-datachannel max-message-size=65536
   a=sctp-port 5000
   a=ice-ufrag:7sFvz2gdLkEwjZEr
   a=ice-pwd:dOTZKZNVlO9RSGsEGM63JXT2
   a=ice-options:trickle
   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
```

The SDP for |candidate-B4| looks like:

```
candidate:109270924 1 udp 2122194687 192.168.2.3 61665 typ host
```

The SDP for |candidate-B5| looks like:

```
   candidate:4036177504 1 udp 1685987071 55.66.77.88 64532 typ srflx
             raddr 192.168.2.3 rport 61665

   The SDP for |candidate-B6| looks like:

   candidate:3671762467 1 udp 41819903 66.77.88.99 50416 typ relay
             raddr 55.66.77.88 rport 64532

   The SDP for |offer-B2| looks like: (note the increment of the version
   number in the o= line, and the c= and a=rtcp lines, which indicate
   the local candidate that was selected)

   v=0
   o=- 7729291447651054566 2 IN IP4 0.0.0.0
   s=-
   t=0 0
   a=msid-semantic:WMS
   a=group:BUNDLE a1 d1 v1 v2
   m=audio 64532 UDP/TLS/RTP/SAVPF 96 0 8 97 98
   c=IN IP4 55.66.77.88
   a=rtcp:64532 IN IP4 55.66.77.88
   a=mid:a1
   a=msid:QI39StLS8W7ZbQl1sJsWUXkr3Zf12fJUvzQ1
          QI39StLS8W7ZbQl1sJsWUXkr3Zf12fJUvzQ1a0
   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=ice-ufrag:7sFvz2gdLkEwjZEr
   a=ice-pwd:dOTZKZNVlO9RSGsEGM63JXT2
   a=ice-options:trickle
   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:actpass
   a=rtcp-mux
   a=rtcp-rsize
   a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level
   a=extmap:2 urn:ietf:params:rtp-hdrext:sdes:mid
   a=ssrc:4429951804 cname:Q/NWs1ao1HmN4Xa5
   a=candidate:109270924 1 udp 2122194687 192.168.2.3 61665 typ host
   a=candidate:4036177504 1 udp 1685987071 55.66.77.88 64532 typ srflx
               raddr 192.168.2.3 rport 61665
   a=candidate:3671762467 1 udp 41819903 66.77.88.99 50416 typ relay
               raddr 55.66.77.88 rport 64532
   a=end-of-candidates
```

```
   m=application 64532 UDP/DTLS/SCTP webrtc-datachannel
   c=IN IP4 55.66.77.88
   a=mid:d1
   a=fmtp:webrtc-datachannel max-message-size=65536
   a=sctp-port 5000
   a=ice-ufrag:7sFvz2gdLkEwjZEr
   a=ice-pwd:dOTZKZNVlO9RSGsEGM63JXT2
   a=ice-options:trickle
   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:actpass
   a=candidate:109270924 1 udp 2122194687 192.168.2.3 61665 typ host
   a=candidate:4036177504 1 udp 1685987071 55.66.77.88 64532 typ srflx
               raddr 192.168.2.3 rport 61665
   a=candidate:3671762467 1 udp 41819903 66.77.88.99 50416 typ relay
               raddr 55.66.77.88 rport 64532
   a=end-of-candidates

   m=video 64532 UDP/TLS/RTP/SAVPF 100 101
   c=IN IP4 55.66.77.88
   a=rtcp:64532 IN IP4 55.66.77.88
   a=mid:v1
   a=msid:61317484-2ed4-49d7-9eb7-1414322a7aae
          f30bdb4a-5db8-49b5-bcdc-e0c9a23172e0
   a=sendrecv
   a=rtpmap:100 VP8/90000
   a=rtpmap:101 rtx/90000
   a=fmtp:101 apt=100
   a=ice-ufrag:7sFvz2gdLkEwjZEr
   a=ice-pwd:dOTZKZNVlO9RSGsEGM63JXT2
   a=ice-options:trickle
   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=rtcp-mux
   a=rtcp-rsize
   a=extmap:2 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=ssrc:1366781083 cname:Q/NWs1ao1HmN4Xa5
   a=ssrc:1366781084 cname:Q/NWs1ao1HmN4Xa5
   a=ssrc-group:FID 1366781083 1366781084
   a=candidate:109270924 1 udp 2122194687 192.168.2.3 61665 typ host
   a=candidate:4036177504 1 udp 1685987071 55.66.77.88 64532 typ srflx
               raddr 192.168.2.3 rport 61665
   a=candidate:3671762467 1 udp 41819903 66.77.88.99 50416 typ relay
```

```
             raddr 55.66.77.88 rport 64532
   a=end-of-candidates

   m=video 64532 UDP/TLS/RTP/SAVPF 100 101
   c=IN IP4 55.66.77.88
   a=rtcp:64532 IN IP4 55.66.77.88
   a=mid:v1
   a=msid:71317484-2ed4-49d7-9eb7-1414322a7aae
          f30bdb4a-5db8-49b5-bcdc-e0c9a23172e0
   a=sendrecv
   a=rtpmap:100 VP8/90000
   a=rtpmap:101 rtx/90000
   a=fmtp:101 apt=100
   a=ice-ufrag:7sFvz2gdLkEwjZEr
   a=ice-pwd:dOTZKZNVlO9RSGsEGM63JXT2
   a=ice-options:trickle
   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=rtcp-mux
   a=rtcp-rsize
   a=extmap:2 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=ssrc:2366781083 cname:Q/NWs1ao1HmN4Xa5
   a=ssrc:2366781084 cname:Q/NWs1ao1HmN4Xa5
   a=ssrc-group:FID 2366781083 2366781084
   a=candidate:109270924 1 udp 2122194687 192.168.2.3 61665 typ host
   a=candidate:4036177504 1 udp 1685987071 55.66.77.88 64532 typ srflx
               raddr 192.168.2.3 rport 61665
   a=candidate:3671762467 1 udp 41819903 66.77.88.99 50416 typ relay
               raddr 55.66.77.88 rport 64532
   a=end-of-candidates


   The SDP for |answer-B2| looks like: (note the use of setup:passive to
   maintain the existing DTLS roles, and the use of a=recvonly to
   indicate that the video streams are one-way)

   v=0
   o=- 4962303333179871723 2 IN IP4 0.0.0.0
   s=-
   t=0 0
   a=msid-semantic:WMS
   a=group:BUNDLE a1 d1 v1 v2
   m=audio 52546 UDP/TLS/RTP/SAVPF 96 0 8 97 98
```

```
   c=IN IP4 11.22.33.44
   a=rtcp:52546 IN IP4 11.22.33.44
   a=mid:a1
   a=msid:57017fee-b6c1-4162-929c-a25110252400
          e83006c5-a0ff-4e0a-9ed9-d3e6747be7d9
   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=ice-ufrag:ATEn1v9DoTMB9J4r
   a=ice-pwd:AtSK0WpNtpUjkY4+86js7ZQl
   a=ice-options:trickle
   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:passive
   a=rtcp-mux
   a=rtcp-rsize
   a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level
   a=extmap:2 urn:ietf:params:rtp-hdrext:sdes:mid
   a=ssrc:1732846380 cname:FocUG1f0fcg/yvY7
   a=candidate:109270923 1 udp 2122194687 192.168.1.2 51556 typ host
   a=candidate:4036177503 1 udp 1685987071 11.22.33.44 52546 typ srflx
               raddr 192.168.1.2 rport 51556
   a=candidate:3671762466 1 udp 41819903 22.33.44.55 61405 typ relay
               raddr 11.22.33.44 rport 52546
   a=end-of-candidates

   m=application 52546 UDP/DTLS/SCTP webrtc-datachannel
   c=IN IP4 11.22.33.44
   a=mid:d1
   a=fmtp:webrtc-datachannel max-message-size=65536
   a=sctp-port 5000
   a=ice-ufrag:ATEn1v9DoTMB9J4r
   a=ice-pwd:AtSK0WpNtpUjkY4+86js7ZQl
   a=ice-options:trickle
   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:passive
   a=candidate:109270923 1 udp 2122194687 192.168.1.2 51556 typ host
   a=candidate:4036177503 1 udp 1685987071 11.22.33.44 52546 typ srflx
               raddr 192.168.1.2 rport 51556
   a=candidate:3671762466 1 udp 41819903 22.33.44.55 61405 typ relay
               raddr 11.22.33.44 rport 52546
   a=end-of-candidates
```

```
   m=video 52546 UDP/TLS/RTP/SAVPF 100 101
   c=IN IP4 11.22.33.44
   a=rtcp:52546 IN IP4 11.22.33.44
   a=mid:v1
   a=recvonly
   a=rtpmap:100 VP8/90000
   a=rtpmap:101 rtx/90000
   a=fmtp:101 apt=100
   a=ice-ufrag:ATEn1v9DoTMB9J4r
   a=ice-pwd:AtSK0WpNtpUjkY4+86js7ZQl
   a=ice-options:trickle
   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:passive
   a=rtcp-mux
   a=rtcp-rsize
   a=extmap:2 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=candidate:109270923 1 udp 2122194687 192.168.1.2 51556 typ host
   a=candidate:4036177503 1 udp 1685987071 11.22.33.44 52546 typ srflx
             raddr 192.168.1.2 rport 51556
   a=candidate:3671762466 1 udp 41819903 22.33.44.55 61405 typ relay
             raddr 11.22.33.44 rport 52546
   a=end-of-candidates

   m=video 52546 UDP/TLS/RTP/SAVPF 100 101
   c=IN IP4 11.22.33.44
   a=rtcp:52546 IN IP4 11.22.33.44
   a=mid:v2
   a=recvonly
   a=rtpmap:100 VP8/90000
   a=rtpmap:101 rtx/90000
   a=fmtp:101 apt=100
   a=ice-ufrag:ATEn1v9DoTMB9J4r
   a=ice-pwd:AtSK0WpNtpUjkY4+86js7ZQl
   a=ice-options:trickle
   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:passive
   a=rtcp-mux
   a=rtcp-rsize
   a=extmap:2 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=candidate:109270923 1 udp 2122194687 192.168.1.2 51556 typ host
a=candidate:4036177503 1 udp 1685987071 11.22.33.44 52546 typ srflx
            raddr 192.168.1.2 rport 51556
a=candidate:3671762466 1 udp 41819903 22.33.44.55 61405 typ relay
            raddr 11.22.33.44 rport 52546
a=end-of-candidates
```

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 browser. 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() (perhaps suitably modified as described in Section 6, there is no guarantee that applications do so. The browser 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 browser 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

Significant text incorporated in the draft as well and review was provided by Harald Alvestrand and Suhas Nandakumar. Dan Burnett, Neil Stratford, Eric Rescorla, Anant Narayanan, Andrew Hutton,

Richard Ejzak, Adam Bergkvist and Matthew Kaufman all provided valuable feedback on this proposal.

11. References

11.1. Normative References

[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-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-trickle-ice]
Ivov, E., Rescorla, E., and J. Uberti, "Trickle ICE: Incremental Provisioning of Candidates for the Interactive Connectivity Establishment (ICE) Protocol", draft-ietf-mmusic-trickle-ice-00 (work in progress), March 2013.

[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-data-protocol]
Jesup, R., Loreto, S., and M. Tuexen, "WebRTC Data Channel Protocol", draft-ietf-rtcweb-data-protocol-04 (work in progress), February 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.

[I-D.nandakumar-mmusic-proto-iana-registration]
Nandakumar, S., "IANA registration of SDP 'proto' attribute for transporting RTP Media over TCP under various RTP profiles.", September 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.

[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.

[RFC5124] Ott, J. and E. Carrara, "Extended Secure RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/SAVPF)", RFC 5124, February 2008.

[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.

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

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

[RFC7022] Begen, A., Perkins, C., Wing, D., and E. Rescorla, "Guidelines for Choosing RTP Control Protocol (RTCP) Canonical Names (CNAMEs)", RFC 7022, September 2013.

11.2. Informative References

[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.

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

[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.

[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.

[RFC5956] Begen, A., "Forward Error Correction Grouping Semantics in the Session Description Protocol", RFC 5956, September 2010.

[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. Change log

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

Changes in draft-09:">

- Don't return null for {local,remote}Description after close().
- Changed TCP/TLS to UDP/DTLS in RTP profile names.
- Separate out bundle and mux policy.
- Added specific references to FEC mechanisms.
- Added canTrickle mechanism.
- Added section on subsequent answers and, answer options.
- Added text defining set{Local,Remote}Description behavior.

Changes in draft-08:

- Added new example section and removed old examples in appendix.
- Fixed <proto> field handling.
- Added text describing a=rtcp attribute.
- Reworked handling of OfferToReceiveAudio and OfferToReceiveVideo per discussion at IETF 90.
- Reworked trickle ICE handling and its impact on m= and c= lines per discussion at interim.
- Added max-bundle-and-rtcp-mux policy.
- Added description of maxptime handling.
- Updated ICE candidate pool default to 0.
- Resolved open issues around AppID/receiver-ID.

- Reworked and expanded how changes to the ICE configuration are handled.
- Some reference updates.
- Editorial clarification.

Changes in draft-07:

- Expanded discussion of VAD and Opus DTX.
- Added a security considerations section.
- Rewrote the section on modifying SDP to require implementations to clearly indicate whether any given modification is allowed.
- Clarified impact of IceRestart on CreateOffer in local-offer state.
- Guidance on whether attributes should be defined at the media level or the session level.
- Renamed "default" bundle policy to "balanced".
- Removed default ICE candidate pool size and clarify how it works.
- Defined a canonical order for assignment of MSTs to m= lines.
- Removed discussion of rehydration.
- Added Eric Rescorla as a draft editor.
- Cleaned up references.
- Editorial cleanup

Changes in draft-06:

- Reworked handling of m= line recycling.
- Added handling of BUNDLE and bundle-only.
- Clarified handling of rollback.
- Added text describing the ICE Candidate Pool and its behavior.
- Allowed OfferToReceiveX to create multiple recvonly m= sections.

Changes in draft-05:

- Fixed several issues identified in the createOffer/Answer sections during document review.
- Updated references.

Changes in draft-04:

- Filled in sections on createOffer and createAnswer.
- Added SDP examples.
- Fixed references.

Changes in draft-03:

- Added text describing relationship to W3C specification

Changes in draft-02:

- Converted from nroff
- Removed comparisons to old approaches abandoned by the working group
- Removed stuff that has moved to W3C specification
- Align SDP handling with W3C draft
- Clarified section on forking.

Changes in draft-01:

- Added diagrams for architecture and state machine.
- Added sections on forking and rehydration.
- Clarified meaning of "pranswer" and "answer".
- Reworked how ICE restarts and media directions are controlled.
- Added list of parameters that can be changed in a description.
- Updated suggested API and examples to match latest thinking.
- 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 Ave S
Kirkland, WA 98033
USA

Email: justin@uberti.name

Cullen Jennings
Cisco
170 West Tasman Drive
San Jose, CA 95134
USA

Email: fluffy@iii.ca

Eric Rescorla (editor)
Mozilla
331 Evelyn Ave
Mountain View, CA 94041
USA

Email: ekr@rtfm.com