Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 18, 2014

P. Kyzivat
L. Xiao
C. Groves
Huawei
R. Hansen
Cisco Systems

February 14, 2014

CLUE Signaling
draft-kyzivat-clue-signaling-07

Abstract

This document specifies how CLUE-specific signaling such as the CLUE protocol [I-D.presta-clue-protocol] and the CLUE data channel [I-D.holmberg-clue-datachannel] are used with each other and with existing signaling mechanisms such as SIP and SDP to produce a telepresence call.

Table of Contents

1. Introduction

To enable devices to participate in a telepresence call, selecting the sources they wish to view, receiving those media sources and displaying them in an optimal fashion, CLUE involves two principal and inter-related protocol negotiations. SDP, conveyed via SIP, is used to negotiate the specific media capabilities that can be delivered to specific addresses on a device. Meanwhile, a CLUE protocol [I-D.presta-clue-protocol], transported via a CLUE data channel [I-D.holmberg-clue-datachannel], is used to negotiate the capture sources available, their attributes and any constraints in their use, along which which captures the far end provides a device wishes to receive.

Beyond negotiating the CLUE channel, SDP is also used to negotiate the details of supported media streams and the maximum capability of each of those streams. As the CLUE Framework [I-D.ietf-clue-framework] defines a manner in which the media provider expresses their maximum encoding capabilities, SDP is also used to express the encoding limits for each potential encoding.

Backwards-compatibility is an important consideration of the document: it is vital that a CLUE-capable device contacting a device that does not support CLUE is able to fall back to a fully functional non-CLUE call. The document also defines how a non-CLUE call may be upgraded to CLUE in mid-call, and similarly how CLUE functionality can be removed mid-call to return to a standard non-CLUE call.

This document originally also defined the CLUE protocol itself. These details have mostly been split out into [I-D.presta-clue-protocol] and expanded, but at present some details remain in this document.

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].

This document draws liberally from the terminology defined in the CLUE Framework [I-D.ietf-clue-framework].

Other terms introduced here:

CLUE data channel: A reliable, bidirectional, transport mechanism used to convey CLUE messages. See [I-D.holmberg-clue-datachannel] for more details..

CLUE-capable device: A device that supports the CLUE data channel [I-D.holmberg-clue-datachannel], the CLUE protocol [I-D.presta-clue-protocol] and the principles of CLUE negotiation.

CLUE-enabled device: A CLUE-capable device that wishes to negotiate a CLUE data channel and send and/or receive CLUe-controlled media.

Non-CLUE device: A device that supports standard SIP and SDP, but either does not support CLUE, or that does but does not currently wish to invoke CLUE capabilities.

CLUE-controlled media: A media "m" line that is under CLUE control; the caprute source that provides the media on this "m" line is negotiated in CLUE. There is a corresponding "non-CLUE-controlled" media term. See Section 4 for details of how this control is signalled in SDP

3. CLUE call establishment

3.1. Establishment of the CLUE data channel

The CLUE data channel [I-D.holmberg-clue-datachannel] is a bidirectional SCTP over DTLS channel used for the transport of CLUE messages. This channel must be established before CLUE protocol messages can be exchanged and CLUE-controlled media can be sent.

Presence of the CLUE data channel in an SDP offer or answer also served as an indication that the device supports CLUE and wishes to upgrade the call to include CLUE-controlled media. A CLUE-enabled device SHOULD include an "m" line for the CLUE channel in its initial SDP offer, and SHOULD include an "m" line in subsequent offers and answers, when allowed by [RFC3264].

In cases where both devices in an SDP negotiation are CLUE-enabled and include an "m" line for the data channel, see [I-D.holmberg-clue-datachannel] for negotiation details. If negotiation is successful, the call is now considered CLUE-enabled, and sending of CLUE protocol [I-D.presta-clue-protocol] messages can begin.

3.2. Initial media transmission

In the event that the CLUE data channel is successfully negotiated, a CLUE-enabled device MAY choose not to send media on the non-CLUE-controlled channels during the period in which control of the CLUE-controlled media lines is negotiated. However, a CLUE-enabled device MUST still be prepared to receive media on non-CLUE-controlled media

lines as defined in [RFC3264].

3.3. Interoperability with non-CLUE devices

A CLUE-enabled device sending an initial SDP offer SHOULD NOT include any "m" line for CLUE-controlled media beyond the "m" line for the CLUE data channel, and SHOULD include at least one non-CLUE-controlled media "m" line.

In the event that the CLUE data channel is not negotiated in the initial offer/answer then CLUE is not in use in the call, and the CLUE-enabled devices MUST either revert to non-CLUE behaviour or terminate the call.

3.4. Mid-call changes to CLUE status

A CLUE-enabled device that receives an initial SDP offer from a non-CLUE device with no CLUE data channel "m" line SHOULD include a new data channel "m" line in any subsequent offers it sends, to indicate that it is CLUE-enabled.

If, in an ongoing non-CLUE call, one or both sides of the call subsequently add the CLUE data channel "m" line to their SDP and the CLUE data channel is then negotiated successfully the call is then considered CLUE-enabled, and sending of CLUE protocol [I-D.presta-clue-protocol] messages can begin.

If, in an ongoing CLUE-enabled call, an SDP offer-answer negotiation completes in a fashion in which the CLUE data channel is no longer active, the call is no longer considered CLUE-enabled. Devices in the call must revert to non-CLUE behaviour or terminate the call.

4. CLUE use of SDP O/A

4.1. Signalling CLUE Encodings

The CLUE Framework [I-D.ietf-clue-framework] defines the concept of "encodings", which represent the sender’s encode ability. Each encoding the media provider wishes to signal is signalled via an "m" line of the appropriate media type, which MUST be marked as sendonly with the "a=sendonly" attribute or as inactive with the "a=inactive" attribute.

The encoder limits of active (eg, "a=sendonly") encodings can then be expressed using existing SDP syntax. For instance, for H.264 see Table 6 in [RFC6184] for a list of valid parameters for representing encoder sender stream limits.

Every "m" line representing a CLUE encoding SHOULD contain a "label" attribute as defined in [RFC4574]. This label is used to identify the encoding by the sender in CLUE ADVERTISEMENT messages and by the receiver in CLUE CONFIGURE messages.

4.1.1. Alternate encoding limit syntaxes

Note that while the expressing of CLUE encoding limits in SDP has been discussed at some length by the working group and it has been agreed that this is the current, working assumption, formal consensus has not been agreed on this. Alternatives include placing encoding limits in the CLUE ADVERTISEMENT message, or by using alternate SDP syntax, such as is suggested in [I-D.groves-clue-latent-config].

4.2. Negotiating receipt of CLUE capture encodings in SDP

A receiver who wishes to receive a CLUE stream via a specific encoding requires an "a=recvonly" "m" line that matches the "a=sendonly" encoding. As well as the normal restrictions defined in [RFC3264] media MUST NOT be sent on this stream until the sender has received a valid CLUE CONFIGURE message specifying the capture to be used for this stream.

4.3. Signaling CLUE control of "m" lines

In many cases an implementation may wish to mix media channels that are under CLUE control with those that are not. It may want to ensure that there are non-CLUE streams for purposes of interoperability, or that can provide media from the start of the call before CLUE negotiation completes, or because the implementation wants CLUE-controlled video but traditional audio, or for any other reasons.

Which "m" lines in an SDP body are under control of the CLUE channel is signalled via the SDP Grouping Framework [RFC5888]. Devices that wish to negotiate CLUE MUST support the grouping framework.

A new semantic for the "group" session-level attribute, "CLUE", is used to signal which "m" lines are under the control of a CLUE channel. As per the framework, all of the "m" lines of a session description that uses "group" MUST be identified with a "mid" attribute whether they are controlled by CLUE or not. The "mid" id of any "m" lines controlled by a CLUE channel MUST be included in the "CLUE" group attribute alongside the "mid" id of the CLUE channel controlling them.

The CLUE group MUST NOT include more than one "m" line for a CLUE data channel. If a CLUE data channel is part of the CLUE group

attribute other media "m" lines included in the group are under the control of that CLUE channel; media MUST NOT be sent or received on these "m" lines until the CLUE channel has been negotiated and media has been negotiated via the CLUE protocol. If no CLUE data channel is part of the CLUE group attribute then media MUST NOT be sent or received on these "m" lines.

"m" lines not specified as under CLUE control follow normal rules for media streams negotiated in SDP as defined in documents such as [RFC3264].

An SDP MAY include more than one group attribute with the "CLUE" semantic. An "mid" id for a given "m" line MUST NOT be included in more than one CLUE group.

4.4. Media line directionality

Presently, this specification mandates that CLUE-controlled "m"-lines must be unidirectional. This is because setting "m"-lines to "a=sendonly" allows the encoder limits to be expressed, whereas in other cases codec attributes express the receive capabilities of a media line.

It is possible that in future versions of this draft or its successor this restriction will be relaxed. If a device does not feel there is a benefit to expressing encode limitations, or if there are no meaningful codec-specific limitations to express (such as with many audio codecs) there are benefits to allowing bidirectional "m"-lines. With bidirectional media lines recipients do not always need to create a new offer to add their own "m"-lines to express their send capabilities; if they can produce an equal or lesser number of streams to send then they may not need additional "m"-lines.

However, at present the need to express encode limitations and the wish to simplify the offer/answer procedure means that for the time being only unidirectional media lines are allowed for CLUE-controlled media. The highly asymmetric nature of CLUE means that the probability of the recipient of the initial offer needing to make their own offer to add additional "m"-lines is significantly higher than it is for most other SIP call scenarios, in which there is a tendancy for both sides to have similar numbers of potential audio and video streams they can send.

4.5. Multiplexing CLUE media lines

There is a desire in many use-cases to be able to multiplex multiple RTP streams onto a single port. However, the syntax for doing this in a CLUE or a generic MMUSIC fashion has not yet been determined.

Because there will always also be a need for non-multiplexed operation, the decision was made to move forward with non-multiplexed syntax, and add multiplexing capabilities when syntax for that has been defined.

5. Interaction of CLUE protocol and SDP negotiations

Information about media streams in CLUE is split between two message types: SDP, which defines media addresses and limits, and the CLUE channel, which defines properties of capture devices available, scene information and additional constraints. As a result certain operations, such as advertising support for a new transmissible capture with associated stream, cannot be performed atomically, as they require changes to both SDP and CLUE messaging.

This section defines how the negotiation of the two protocols interact, provides some recommendations on dealing with intermediary stages in non-atomic operations, and mandates additional constraints on when CLUE-configured media can be sent.

5.1. Independence of SDP and CLUE negotiation

To avoid complicated state machines with the potential to reach invalid states if messages were to be lost, or be rewritten en-route by middle boxes, the current proposal is that SDP and CLUE messages are independent. The state of the CLUE channel does not restrict when an implementation may send a new SDP offer or answer, and likewise the implementation’s ability to send a new CLUE ADVERTISEMENT or CONFIGURE message is not restricted by the results of or the state of the most recent SDP negotiation.

The primary implication of this is that a device may receive an SDP with a CLUE encoding it does not yet have capture information for, or receive a CLUE CONFIGURE message specifying a capture encoding for which the far end has not negotiated a media stream in SDP.

CLUE messages contain an EncodingID which is used to identify a specific encoding in SDP. The non-atomic nature of CLUE negotiation means that a sender may wish to send a new ADVERTISEMENT before the corresponding SDP message. As such the sender of the CLUE message MAY include an EncodingID which does not currently match an extant id in SDP.

5.2. Recommendations for operating with non-atomic operations

Generally, implementations that receive messages for which they have incomplete information SHOULD wait until they have the corresponding

information they lack before sending messages to make changes related to that information. For instance, an implementation that receives a new SDP offer with three new "a=sendonly" CLUE "m" lines that has not received the corresponding CLUE ADVERTISEMENT providing the capture information for those streams SHOULD NOT include corresponding "a=recvonly" lines in its answer, but instead should make a new SDP offer when and if a new ADVERTISEMENT arrives with captures relevant to those encodings.

Because of the constraints of offer/answer and because new SDP negotiations are generally more 'costly' than sending a new CLUE message, implementations needing to make changes to both channels SHOULD prioritize sending the updated CLUE message over sending the new SDP message. The aim is for the recipient to receive the CLUE changes before the SDP changes, allowing the recipient to send their SDP answers without incomplete information, reducing the number of new SDP offers required.

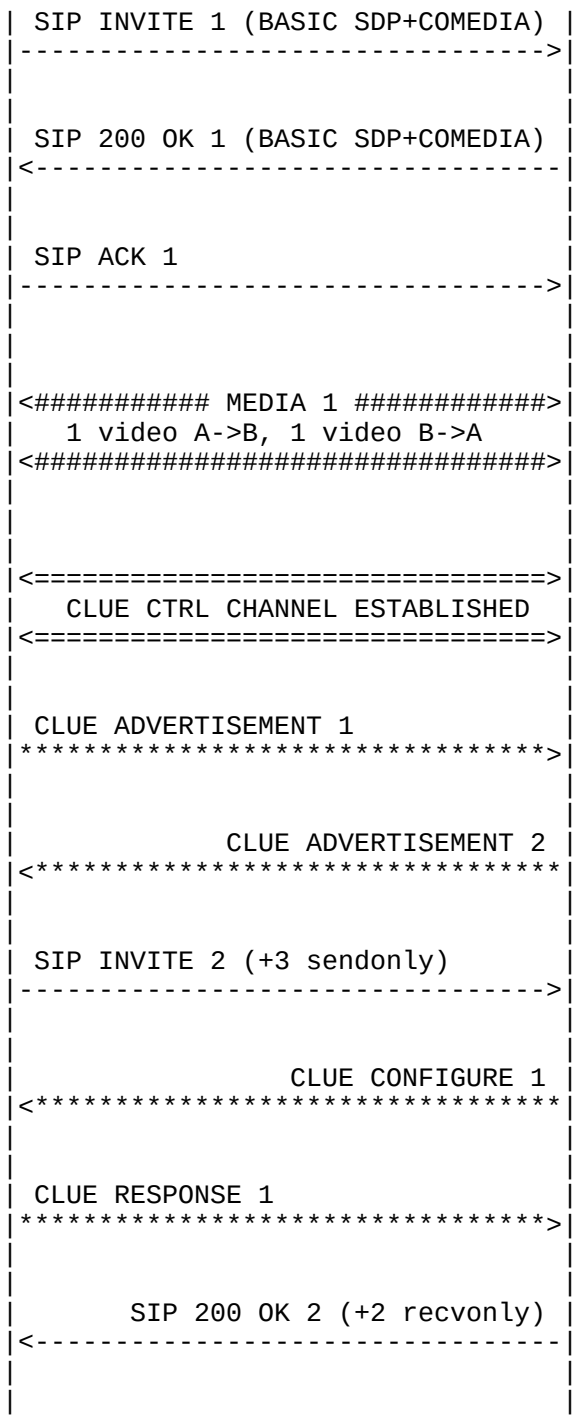
5.3. Constraints on sending media

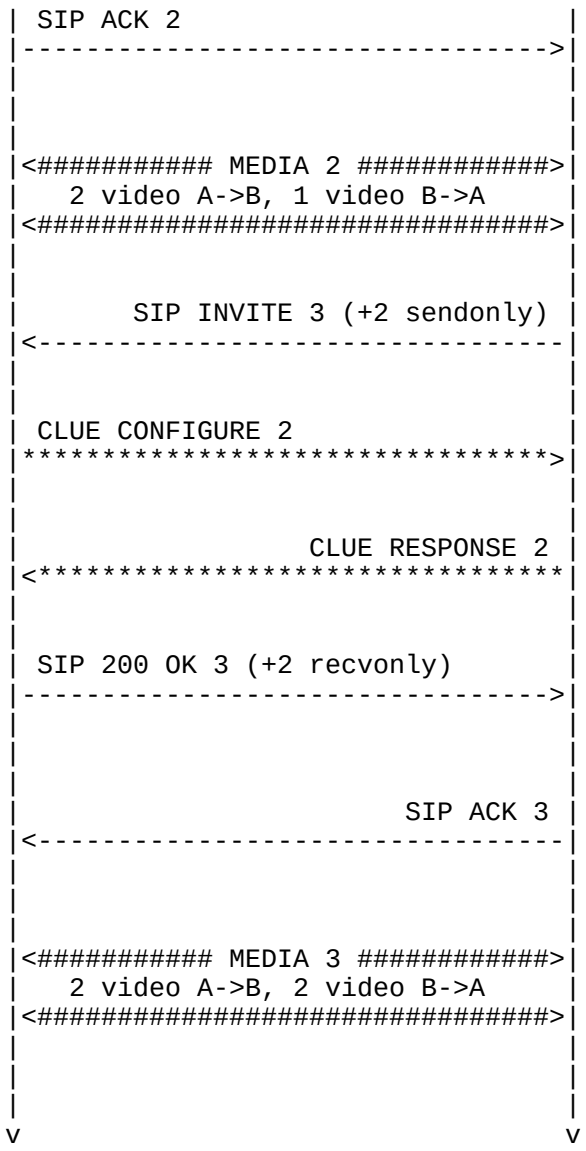
While SDP and CLUE message states do not impose constraints on each other, both impose constraints on the sending of media - media MUST NOT be sent unless it has been negotiated in both CLUE and SDP: an implementation MUST NOT send a specific CLUE capture encoding unless its most recent SDP exchange contains an active media channel for that encoding AND the far end has sent a CLUE CONFIGURE message specifying a valid capture for that encoding.

6. Example: A call between two CLUE-capable endpoints

This example illustrates a call between two CLUE-capable endpoints. Alice, initiating the call, is a system with three cameras and three screens. Bob, receiving the call, is a system with two cameras and two screens. A call-flow diagram is presented, followed by an summary of each message.

To manage the size of this section only video is considered, and SDP snippets only illustrate video 'm' lines. ACKs are not discussed.


SIP INVITE 1 (BASIC SDP+COMEDIA)
SIP 200 OK 1 (BASIC SDP+COMEDIA)
SIP ACK 1
MEDIA 1
1 video A->B, 1 video B->A
CLUE CTRL CHANNEL ESTABLISHED
CLUE ADVERTISEMENT 1
CLUE ADVERTISEMENT 2
SIP INVITE 2 (+3 sendonly)
CLUE CONFIGURE 1
CLUE RESPONSE 1
SIP 200 OK 2 (+2 recvonly)



In INVITE 1, Alice sends Bob a SIP INVITE including in the SDP body the basilar audio and video capabilities ("BASIC SDP") and the information needed for opening a control channel to be used for CLUE protocol messages exchange, according to what is envisioned in the COMEDIA approach ("COMEDIA") for DTLS/SCTP channel [I-D.ietf-mmusic-sctp-sdp]. A snippet of the SDP showing the grouping attribute and the video m-line are shown below (mid 3

represents the CLUE channel):

```
...
a=group:CLUE 3
...
m=video 6002 RTP/AVP 96
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=42e016;max-mbps=108000;max-fs=3600
a=sendrecv
a=mid:2
```

Bob responds with a similar SDP (200 OK 1); due to their similiarity no SDP snippet is shown here. Alice and Bob are each able to send a single audio and video stream (whether they choose to send this initial media before CLUE has been negotiated is implementation-dependent). This is illustrated as MEDIA 1.

With the successful initial O/A Alice and Bob are also free to negotiate the CLUE channel. Once this is successfully established CLUE negotiation can begin. This is illustrated as CLUE CHANNEL ESTABLISHED.

Alice now sends her CLUE Advertisement (ADVERTISEMENT 1). She advertises three static captures representing her three cameras. She also includes switched captures suitable for two- and one-screen systems. All of these captures are in a single capture scene, with suitable capture scene entries to tell Bob that he should either subscribe to the three static captures, the two switched capture view or the one switched capture view. Alice has no simultaneity constraints, so includes all six captures in one simultaneous set. Finally, Alice includes an encoding group with three encoding IDs: "enc1", "enc2" and "enc3". These encoding ids aren’t currently valid, but will match the next SDP offer she sends.

Bob received ADVERTISEMENT 1 but does not yet send a Configure message, because he has not yet received Alice’s encoding information, so as yet he does not know if she will have sufficient resources to send him the two streams he ideally wants at a quality he is happy with.

Bob also sends his CLUE ADVERTISEMENT (ADVERTISEMENT 2). He advertises two static captures representing his cameras. He also includes a single composed capture for single-screen systems, in which he will composite the two camera views into a single video stream. All three captures are in a single capture scene, with suitable capture scene entries to tell Alice that she should either

subscribe to the two static captures, or the single composed capture. Bob also has no simultaneity constraints, so includes all three captures in one simultaneous set. Bob also includes a single encoding group with two encoding IDs: "foo" and "bar".

Similarly, Alices receives ADVERTISEMENT 2 but does not yet send a CONFIGURE message, because she has not yet received Bob’s encoding information.

Alice now sends INVITE 2. She maintains the sendrecv audio, video and CLUE m-lines, and she adds three new sendonly m-lines to represents the maximum three encodings she can send. Each of these m-lines has a label corresponding to one of the encoding ids from ADVERTISEMENT 1. Each also has its mid added to the grouping attribute to show they are controlled by the CLUE channel. A snippet of the SDP showing the grouping attribute and the video m-lines are shown below (mid 3 represents the CLUE channel):

```
...
a=group:CLUE 3 4 5 6
...
m=video 6002 RTP/AVP 96
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=42e016;max-mbps=108000;max-fs=3600
a=sendrecv
a=mid:2
...
m=video 6004 RTP/AVP 96
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=42e016
a=sendonly
a=mid:4
a=label:enc1
m=video 6006 RTP/AVP 96
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=42e016
a=sendonly
a=mid:5
a=label:enc2
m=video 6008 RTP/AVP 96
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=42e016
a=sendonly
a=mid:6
a=label:enc3
```

Bob now has all the information he needs to decide which streams to configure. As such he now sends CONFIGURE 1. This requests the pair of switched captures that represent Alice's scene, and he configures them with encoder ids "enc1" and "enc2". This also serves as an ack for Alice's ADVERTISMENT 1.

Alice receives Bob's message CONFIGURE 1 and sends RESPONSE 1 to ack its receptions. She does not yet send the capture encodings specified, because at this stage Bob hasn't negotiated the ability to receive these streams in SDP.

Bob now sends his SDP answer as part of 200 OK 2. Alongside his original audio, video and CLUE m-lines he includes two active recvonly m-lines and a zeroed m-line for the third. He adds their mid values to the grouping attribute to show they are controlled by the CLUE channel. A snippet of the SDP showing the grouping attribute and the video m-lines are shown below (mid 100 represents the CLUE channel):

```
...
a=group:CLUE 11 12 100
...
m=video 58722 RTP/AVP 96
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=42e016;max-mbps=108000;max-fs=3600
a=sendrecv
a=mid:10
...
m=video 58724 RTP/AVP 96
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=42e016;max-mbps=108000;max-fs=3600
a=recvonly
a=mid:11
m=video 58726 RTP/AVP 96
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=42e016;max-mbps=108000;max-fs=3600
a=recvonly
a=mid:12
m=video 0 RTP/AVP 96
```

On receiving 200 OK 2 from Bob Alice is now able to send the two streams of video Bob requested - this is illustrated as MEDIA 2.

The constraints of offer/answer meant that Bob could not include his encoder information as new m-lines in 200 OK 2. As such Bob now sends INVITE 3 to generate a new offer. Along with all the streams

from 200 OK 2 Bob also includes two new sendonly streams. Each stream has a label corresponding to the encoding ids in his ADVERTISEMENT 2 message. He also adds their mid values to the grouping attribute to show they are controlled by the CLUE channel. A snippet of the SDP showing the grouping attribute and the video m-lines are shown below (mid 100 represents the CLUE channel):

```
...
a=group:CLUE 11 12 13 14 100
...
m=video 58722 RTP/AVP 96
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=42e016;max-mbps=108000;max-fs=3600
a=sendrecv
a=mid:10
...
m=video 58724 RTP/AVP 96
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=42e016;max-mbps=108000;max-fs=3600
a=recvonly
a=mid:11
m=video 58726 RTP/AVP 96
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=42e016;max-mbps=108000;max-fs=3600
a=recvonly
a=mid:12
m=video 0 RTP/AVP 96
m=video 58728 RTP/AVP 96
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=42e016
a=sendonly
a=label:foo
a=mid:13
m=video 58730 RTP/AVP 96
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=42e016
a=sendonly
a=label:bar
a=mid:14
```

Having received this Alice now has all the information she needs to send CONFIGURE 2. She requests the two static captures from Bob, to be sent on encodings "foo" and "bar".

Bob receives Alice’s message CONFIGURE 2 and sends RESPONSE 2 to ack its receptions. Bob does not yet send the capture encodings

specified, because Alice hasn't yet negotiated the ability to receive these streams in SDP.

Alice now sends 200 OK 3, matching two recvonly m-lines to Bob's new sendonly lines. She includes their mid values in the grouping attribute to show they are controlled by the CLUE channel. A snippet of the SDP showing the grouping attribute and the video m-lines are shown below (mid 3 represents the CLUE channel):

```
...
a=group:CLUE 3 4 5 7 8
...
m=video 6002 RTP/AVP 96
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=42e016;max-mbps=108000;max-fs=3600
a=sendrecv
a=mid:2
...
m=video 6004 RTP/AVP 96
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=42e016
a=sendonly
a=mid:4
a=label:enc1
m=video 6006 RTP/AVP 96
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=42e016
a=sendonly
a=mid:5
a=label:enc2
m=video 0 RTP/AVP 96
m=video 6010 RTP/AVP 96
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=42e016;max-mbps=108000;max-fs=3600
a=recvonly
a=mid:7
m=video 6012 RTP/AVP 96
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=42e016;max-mbps=108000;max-fs=3600
a=recvonly
a=mid:8
```

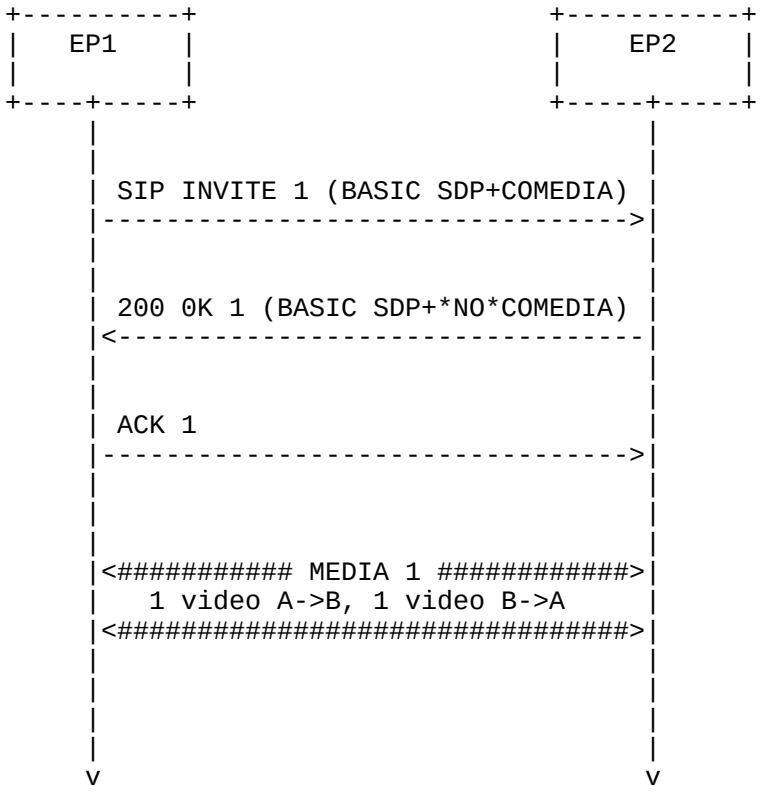
Finally, on receiving 200 OK 3 Bob is now able to send the two streams of video Alice requested - this is illustrated as MEDIA 3.

Both sides of the call are now sending multiple video streams with

their sources defined via CLUE negotiation. As the call progresses either side can send new ADVERTISEMENT or CONFIGURE or new SDP negotiation to add, remove or change what they have available or want to receive.

7. Example: A call between a CLUE-capable and non-CLUE endpoint

In this brief example Alice is a CLUE-capable endpoint making a call to Bob, who is not CLUE-capable, i.e., it is not able to use the CLUE protocol.



In INVITE 1, Alice sends Bob a SIP INVITE including in the SDP body the basilar audio and video capabilities ("BASIC SDP") and the information needed for opening a control channel to be used for CLUE protocol messages exchange, according to what is envisioned in the COMEDIA approach ("COMEDIA") for DTLS/SCTP channel

[I-D.ietf-mmusic-sctp-sdp]. A snippet of the SDP showing the grouping attribute and the video m-line are shown below (mid 3 represents the CLUE channel):

```
...
a=group:CLUE 3
...
m=video 6002 RTP/AVP 96
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=42e016;max-mbps=108000;max-fs=3600
a=sendrecv
a=mid:2
```

Bob is not CLUE capable, and hence does not recognize the "CLUE" semantic for the grouping attribute, not does he support the CLUE channel. He responds with an answer with audio and video, but with the CLUE channel zeroed.

From the lack of the CLUE channel Alice understands that Bob does not support CLUE, or does not wish to use it. Both sides are now able to send a single audio and video stream to each other. Alice at this point begins to send her fallback video: in this case likely a switched view from whichever camera shows the current loudest participant on her side.

8. CLUE requirements on SDP O/A

The current proposal calls for a new "CLUE" semantic for the SDP Grouping Framework [RFC5888].

Any other SDP extensions required to support CLUE signaling should also be specified here. Then we will need to take action within MMUSIC to make those happen. This section should be empty and removed before this document becomes an RFC.

NOTE: The RTP mapping document [I-D.even-clue-rtp-mapping] is also likely to call for SDP extensions. We will have to reconcile how to coordinate these two documents.

9. SIP Signaling

(Placeholder) This may be unremarkable. If so we can drop it.

10. CLUE over RTCWEB

We may want to rule this out of scope for now. But we should be thinking about this.

11. Open Issues

Here are issues pertinent to signaling that need resolution. Resolution will probably result in changes somewhere in this document, but may also impact other documents.

- While the preference is to multiplex multiple capture encodings over a single RTP session, this will not always be desirable or possible. The factors that prevent multiplexing may come from either the provider or the consumer. So the extent of multiplexing must be negotiated. The decision about how to multiplex affects the number and grouping of m-lines in the SDP. The endpoint of a CLUE session that sends an offer needs to know the mapping of capture encodings to m-lines for both sides.

  AFAIK this issue hasn’t yet been considered at all.
- The current method for expressing encodings in SDP limits the parameters available when describing H264 encoder capabilities to those defined in Table 6 in [RFC6184]

12. What else?

13. Acknowledgements

The team focusing on this draft consists of: Roni Even, Rob Hansen, Christer Holmberg, Paul Kyzivat, Simon Pietro-Romano, Roberta Presta.

Christian Groves has contributed detailed comments and suggestions.

The author list should be updated as people contribute substantial text to this document.

14. IANA Considerations

TBD

15. Security Considerations

TBD

16. Change History

-07: Revisions by Rob Hansen
- Removed the text providing arguments for encoding limits being in SDP and encoding groups in the CLUE protocol in favor of the specifics of how to negotiate encodings in SDP
- Added normative language on the setting up of a CLUE call, and added sections on mid-call changes to the CLUE status.
- Added references to [I-D.holmberg-clue-datachannel] where appropriate.
- Added some terminology for various types of CLUE and non-CLUE states of operation.
- Moved language related to topics that should be in [I-D.holmberg-clue-datachannel] and [I-D.presta-clue-protocol], but that has not yet been resolved in those documents, into an appendix.

-06: Revisions by Rob Hansen
- Removed CLUE message XML schema and details that are now in draft-presta-clue-protocol
- Encoding limits in SDP section updated to note that this has been investigated and discussed and is the current working assumption of the WG, though consensus has not been fully achieved.
- A section has also been added on the current mandation of unidirectional "m"-lines.
- Updated CLUE messaging in example call flow to match draft-presta-clue-protocol-03

-05: Revisions by pkyzivat:
- Specified versioning model and mechanism.
- Added explicit response to all messages.
- Rearranged text to work with the above changes. (Which rendered diff almost useless.)

-04: Revisions by Rob Hansen: ???

-03: Revisions by pkyzivat:
- Added a syntax section with an XML schema for CLUE messages. This is a strawhorse, and is very incomplete, but it establishes a template for doing this based on elements defined in the data model. (Thanks to Roberta for help with this!)
- Did some rewording to fit the syntax section in and reference it.
- Did some relatively minor restructuring of the document to make it flow better in a logical way.

-02: A bunch of revisions by pkyzivat:
- Moved roberta’s call flows to a more appropriate place in the document.
- New section on versioning.

* New section on NAK.
* A couple of possible alternatives for message acknowledgment.
* Some discussion of when/how to signal changes in provider state.
* Some discussion about the handling of transport errors.
* Added a change history section.

These were developed by Lennard Xiao, Christian Groves and Paul, so added Lennard and Christian as authors.

-01: Updated by roberta to include some sample call flows.

-00: Initial version by pkyzivat. Established general outline for the document, and specified a few things thought to represent wg consensus.

17. References

17.1. Normative References

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

[I-D.ietf-clue-framework] Duckworth, M., Pepperell, A., and S. Wenger, "Framework for Telepresence Multi-Streams", draft-ietf-clue-framework-14 (work in progress), February 2014.

[I-D.presta-clue-data-model-schema] Presta, R. and S. Romano, "An XML Schema for the CLUE data model", draft-presta-clue-data-model-schema-03 (work in progress), March 2013.

[I-D.presta-clue-protocol] Presta, R. and S. Romano, "CLUE protocol", draft-presta-clue-protocol-03 (work in progress), November 2013.

[I-D.holmberg-clue-datachannel] Holmberg, C., "CLUE Protocol Data Channel", draft-holmberg-clue-datachannel-03 (work in progress), February 2014.

[I-D.groves-clue-latent-config] Groves, C., Yang, W., and R. Even, "CLUE and latent configurations", draft-groves-clue-latent-config-00 (work in progress), January 2014.

[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-06 (work in progress), February 2014.

[I-D.tuexen-tsvwg-sctp-dtls-encaps]
Jesup, R., Loreto, S., Stewart, R., and M. Tuexen, "DTLS Encapsulation of SCTP Packets for RTCWEB", draft-tuexen-tsvwg-sctp-dtls-encaps-01 (work in progress), July 2012.

[RFC4574] Levin, O. and G. Camarillo, "The Session Description Protocol (SDP) Label Attribute", RFC 4574, August 2006.

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

17.2. Informative References

[RFC4353] Rosenberg, J., "A Framework for Conferencing with the Session Initiation Protocol (SIP)", RFC 4353, February 2006.

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

[RFC6120] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, March 2011.

[RFC6184] Wang, Y., Even, R., Kristensen, T., and R. Jesup, "RTP Payload Format for H.264 Video", RFC 6184, May 2011.

[I-D.even-clue-sdp-clue-relation]
Even, R., "Signalling of CLUE and SDP offer/answer", draft-even-clue-sdp-clue-relation-01 (work in progress), October 2012.

[I-D.even-clue-rtp-mapping]
Even, R. and J. Lennox, "Mapping RTP streams to CLUE media captures", draft-even-clue-rtp-mapping-05 (work in progress), February 2013.

[I-D.hansen-clue-sdp-interaction]
Hansen, R., "SDP and CLUE message interactions", draft-hansen-clue-sdp-interaction-01 (work in progress), February 2013.

Appendix A. CLUE Signalling and data channel concerns

[The specifics of the CLUE signaling protocol are in the process of being defined in [I-D.presta-clue-protocol], while the negotiation of the CLUE data channel is being defined in [I-D.holmberg-clue-datachannel]. As such, considerable text originally in this section have been transitioned to these document. The following text relates to issues that are no longer the focus of this document, but remain important and unresolved, and so have been preserved here.]

A.1. Protocol Versioning and Options

A.1.1. Versioning Objectives

The CLUE versioning mechanism addresses the following needs:

- Coverage:
  - Versioning of basic behavior and options,
  - CLUE message exchange,
  - CLUE message exchange,
  - coordinated use of SIP and SDP,
  - required media behavior.
- Remain fixed for the duration of the CLUE channel
- Be extensible for configuration of new options.
- Be sufficient (with extensions) for all envisioned future versions.

A.1.2. Versioning Overview

An initial message exchange on the CLUE channel handles the negotiation of version and options.

- Dedicated message types are used for this negotiation.
- The negotiation is repeated if the CLUE channel is reestablished.

The version usage is similar in philosophy to XMPP:

- See [RFC6120] section 4.7.5.
- A version has major and minor components. (Each a non-negative integer.)
- Major version changes denote non-interoperable changes.
- Minor version changes denote schema changes that are backward compatible by ignoring unknown XML elements, or other backward compatible changes.
- If a common major version cannot be negotiated, then CLUE MUST NOT be used.

- The same message exchange also negotiates options.
- Each option is denoted by a unique XML element in the negotiation.

Figure 1 shows the negotiation in simplified form:

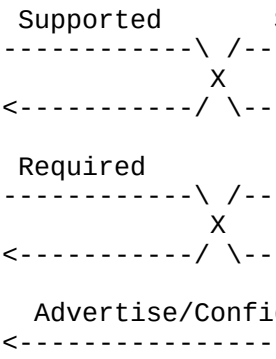


Figure 1: Basic Option Negotiation (simplified)

Dedicated message types are used for the negotiation because:

- The protocol can then ensure that the negotiation is done first, and once. Not changing mid-session means an endpoint can plan ahead, and predict what may be used and what might be received.
- This provides extensible framework for negotiating optional features.
- A full option negotiation can be completed before other messages are exchanged.

Figure 2 and Figure 3 are simplified examples of the Supported and Required messages:

```
<supported>
        <version major="1" minor="0">
          <!- May repeat version if multiple
         major versions supported.      ->
   <!- Options follow ->
   <mediaProvider/>
        ...
</supported>
```

Figure 2: Supported Message (simplified)

```
<required>
        <version major="1" minor="0">
          <!- Requested options of peer follow ->
   <!- Options follow ->
   <mediaProvider/>
        ...
</required>
```

Figure 3: Required Message (simplified)

A.1.3. Version Negotiation

The Supported message includes one or more <version> elements, each denoting a major/minor version combination that the sender of the message is capable of supporting.

The <version> element contains both a major and minor version. Each is a non-negative integer. Each <version> element in the message MUST contain a unique major version number, distinct from the major version number in all the other <version> elements in the message. The minor version in a <version> element denotes the largest minor version the sender supports for the corresponding major version. (Minor versions are always backwards compatible, so support for a minor version implies support for all smaller minor versions.)

Each endpoint of the CLUE channel sends a Supported message, and receives the Supported message sent by the other end. Then each end compares the versions sent and the versions received to determine the version to be used for this CLUE session.

- If there is no major version in common between the two ends, negotiation fails.
- The <version> elements from the two ends that have the largest matching major version are selected.
- After exchange each end determines compatible version numbers to be used for encoding and decoding messages, and other behavior in the CLUE session.
  - The <version> elements from the two ends that have the largest matching major version are selected.
  - The side that sent the smaller minor version chooses the one it sent.
  - The side that sent the larger minor version may choose the minor version it received, or the one it sent, or any value between those two.
- Each end then sends a Required message with a single <version> element containing the major and minor versions it has chosen.

  [[Note: "required" is the wrong semantic for this. Might want a

better message name.]]

- o Each end then behaves in accord with the specifications denoted by the version it chose. This continues until the end of the CLUE session, or until changed as a result of another version negotiation when the CLUE channel is reestablished.

  [[Note: The version negotiation remains in effect even if the CLUE channel is lost.]]

A.1.4. Option Negotiation

Option negotiation is used to agree upon which options will be available for use within the CLUE session. (It does not say that these options must be used.) This may be used for both standard and proprietary options. (As used here, and option could be either a feature described as part of this specification that is optional to implement, or a feature defined in a separate specification that extends this one.)

Each end includes, within the Supported message it sends, elements describing those options it is willing and able to use with this CLUE session.

Each side, upon receiving a Supported message, selects from that message those option elements that it wishes the peer to use. (If/ when occasion for that use arises.) It then includes those selected elements into the Required message that it sends.

Within a received Supported message, unknown option elements MUST be ignored. This includes elements that are of a known type that is not known to denote an option.

A.1.5. Option Elements

Each option is denoted, in the Supported and Required messages, by an XML element. There are no special rules for these elements - they can be any XML element. The attributes and body of the element may carry further information about the option. The same element type is used to denote the option in the Supported message and the corresponding Required message, but the attributes and body may differ according to option-specific rules. This may be used to negotiate aspects of a particular option. The ordering of option elements is irrelevant within the Supported and Required messages, and need not be consistent in the two.

Only one option element is defined in this document: <mediaProvider>.

A.1.5.1. <mediaProvider>

The <mediaProvider> element, when placed in a Supported message, indicates that the sender is willing and able to send ADVERTISEMENT messages and receive CONFIGURE messages. When placed in a Required message, the <mediaProvider> element indicates that the sender is willing, able, and desirous of receiving ADVERTISEMENT messages and sending CONFIGURE messages. If an endpoint does not receive <mediaProvider> in a Required message, it MUST NOT send ADVERTISEMENT messages. For common cases <mediaProvider> should be supported and required by both endpoints, to enable bidirectional exchange of media. If not required by either end, the CLUE session is useless. This is an error condition, and SHOULD result in termination of the CLUE channel.

The <mediaProvider> element has no defined attributes or body.

A.1.6. Version & option negotiation errors

The following are errors that may be detected and reported during version negotiation:

- Version incompatibility

  There is no common value between the major version numbers sent in a Supported message and those in the received Supported message.

- Option incompatibility

  This can occur if options supported by one endpoint are inconsistent with those supported by the other endpoint. E.g., The <mediaProvider> option is not specified by either endpoint. Options SHOULD be specified so as to make it difficult for this problem to occur.

  This error may also be used to indicate that insufficient options have been required among the two ends for a useful session to result. This can occur with a feature that needs to be present on at least one end, but not on a specific end. E.g., The <mediaProvider> option was Supported by at least one of the endpoints, but it was not Required by either.

  This may also be used to indicate that an option element in the Required message has attributes or body content that is syntactically correct, but in inconsistent with the rules for option negotiation specified for that particular element. The definition of each option must specify the negotiation rules for that option.

- Unsupported option

  An option element type received in a Required message did not appear in the corresponding Supported element.

  (Unsupported options received in a Supported message do not trigger this error. They are ignored.)

These errors are reported using the normal message error reporting mechanism.

Other applicable error codes may also be returned in response to a Supported or Required message.

Errors that occur at this stage result in negotiation failure. When this occurs, CLUE cannot be used until the end of the SIP session, or until a new CLUE channel is negotiated and a subsequent version negotiation succeeds. The SIP session may continue without CLUE features.

A.1.7. Definition and Use of Version Numbers

[[NOTE: THIS IS AWKWARD. SUGGESTIONS FOR BETTER WAYS TO DEFINE THIS ARE WELCOME.]]

This document defines CLUE version 1.0 (major=1, minor=0). This denotes the normative behavior defined in this document and other documents upon which it normatively depends, including but is not limited to:

- the schema defined in [I-D.presta-clue-protocol];
- the schema defined in [clue-data-model];
- the protocol used to exchange CLUE messages;
- the protocol defined herein that defines valid sequence of CLUE messages;
- the specific rules defined herein for employing SIP, SDP, and RTP to realize the CLUE messages.

Given two CLUE versions Vx and Vy, then Vx is backward compatible with Vy if and only if:

- All messages valid according to the schema of Vx are also valid according to the schemas of Vy
- All messages valid according to the schema of Vy can be made valid according to the schemas of Vx by deleting elements undefined in the schemas of Vx.

  [[NOTE: THIS PROBABLY NEEDS WORK!]]

- o All normative behaviors defined for Vx are defined consistently for Vy.

  [[NOTE: SOME HAND WAVING HERE.]]

Revisions, updates, to any of the documents denoted by Version 1.0 MAY result in the definition of a new CLUE version. If they do, then this document MUST be revised to define the new version.

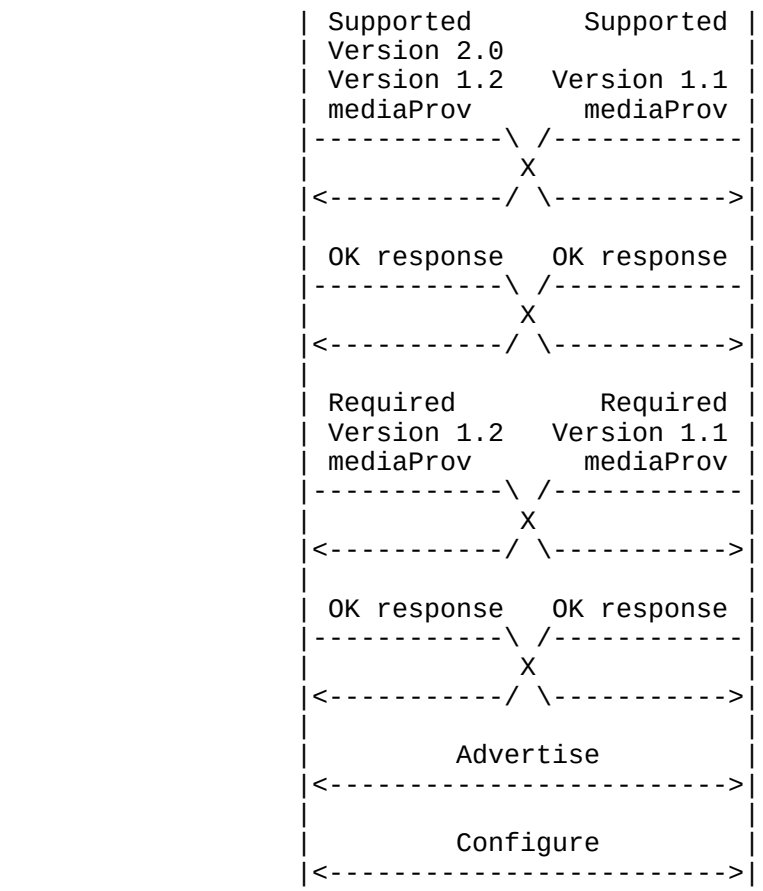
The CLUE version to be defined in a revision to this document MUST be determined as follows:

- o If the revision and the document being revised are mutually backward compatible (they are functionally equivalent), then the CLUE version MUST remain unchanged.
- o Else if the revision is backward compatible with the document being revised, then the CLUE major version MUST remain unchanged, and the CLUE minor version MUST be increased by one (1).
- o Else the CLUE major version must be increased by one (1), and the CLUE minor version set to zero (0).

When a CLUE implementation sends a Supported message, it MUST include the CLUE versions it is willing and able to conform with.

A.1.8. Version & Option Negotiation Examples

A.1.8.1. Successful Negotiation - Multi-version



The endpoint on the left can support versions 1.2 and 2.0, and because of backward compatibility can support versions 1.0 and 1.1. The endpoint on the right supports only version 2.0. Both endpoints with to both provide and consume media. They each send a Supported message indicating what they support.

The element on the left, upon receiving the Supported message, determines that it is permitted to use version 1.2 or 1.1, and decides to use 1.2. It sends a Required message containing version 1.2 and also includes the mediaProvider option element, because it wants its peer to provide media.

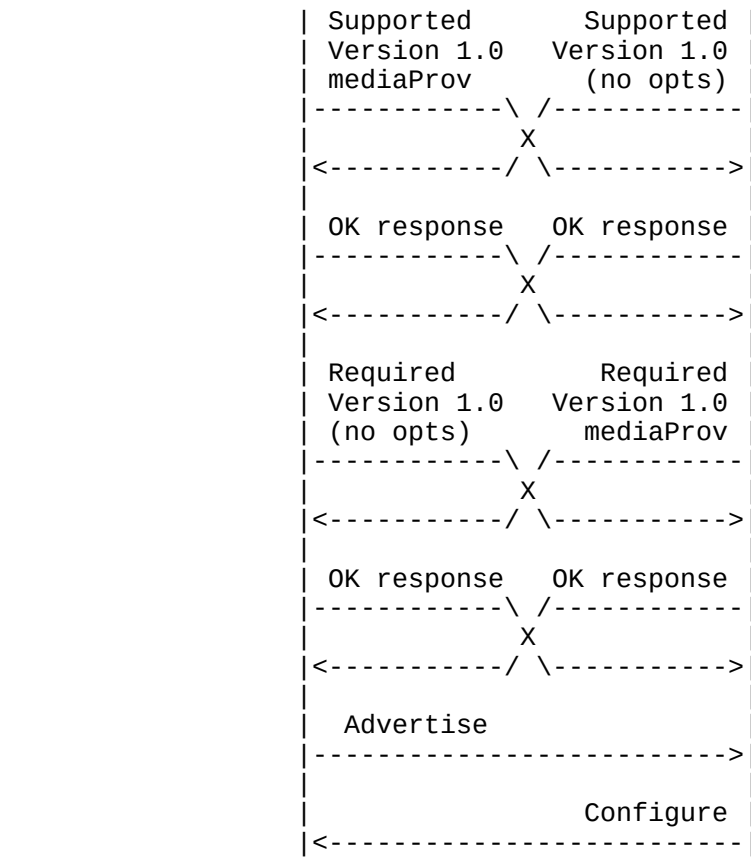
The element on the right, upon receiving the Supported message, selects version 1.1 because it is the highest version in common to the two sides. It sends a Required message containing version 1.1 because that is the highest version in common. It also includes the mediaProvider option element, because it wants its peer to provide

media.

Upon receiving the Required messages, both endpoints determine that they should send ADVERTISEMENTs.

ADVERTISEMENT and CONFIGURE messages will flow in both directions.

A.1.8.2. Successful Negotiation - Consumer-Only Endpoint

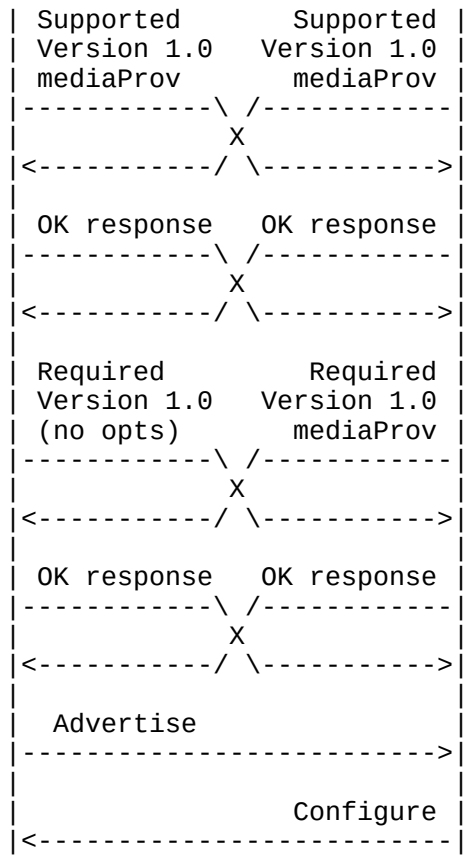


The endpoint on the right consumes media, but doesn’t provide any so it doesn’t include the mediaProvider option element in the Supported message it sends.

The element on the left would like to include a mediaProvider option element in the Requirements message it sends, but can’t because it did not receive one in the Supported message it received.

ADVERTISEMENT messages will only go from left to right, and CONFIGURE messages will only go from right to left.

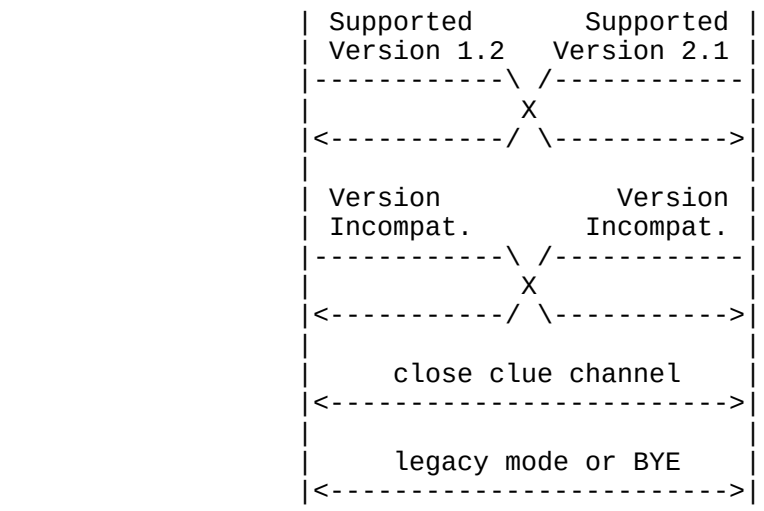
A.1.8.3. Successful Negotiation - Provider-Only Endpoint



The endpoint on the left provides media but does not consume any so it includes the mediaProvider option element in the Supported message it sends, but does't include the mediaProvider option element in the Required message it sends.

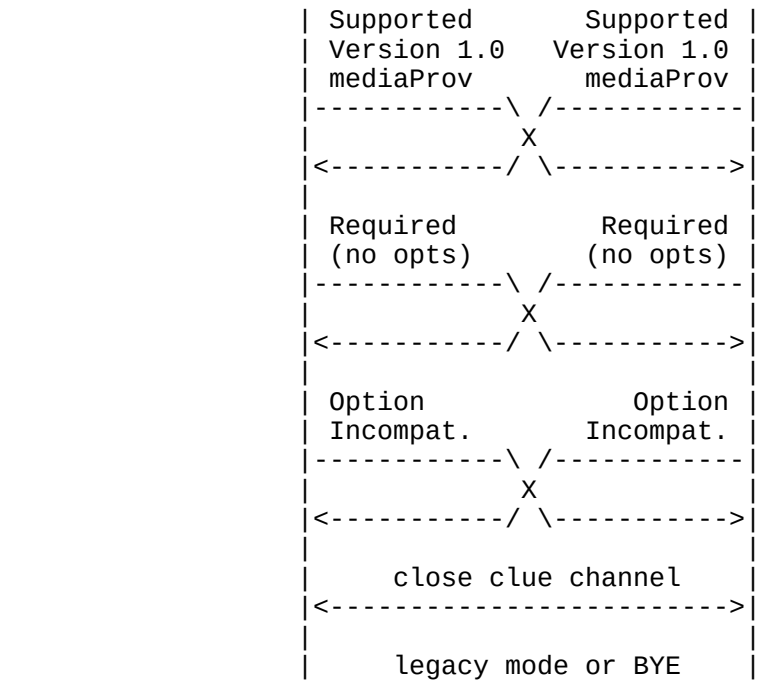
ADVERTISEMENT messages will only go from left to right, and CONFIGURE messages will only go from right to left.

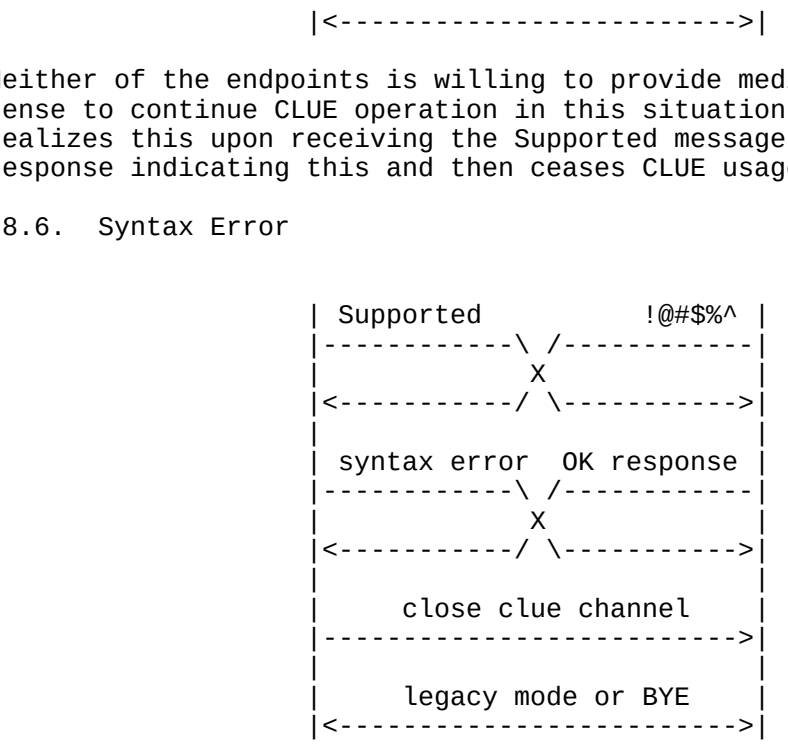
A.1.8.4. Version Incompatibility



Upon receiving the Supported message, each endpoint discovers there is no major version in common, so CLUE usage is not possible. Each sends an error response indicating this and then ceases CLUE usage.

A.1.8.5. Option Incompatibility





Neither of the endpoints is willing to provide media. It makes no sense to continue CLUE operation in this situation. Each endpoint realizes this upon receiving the Supported message, sends an error response indicating this and then ceases CLUE usage.

A.1.8.6. Syntax Error

A.2. Message Transport

CLUE messages are transported over a bidirectional CLUE channel. In a two-party CLUE session, a CLUE channel connects the two endpoints. In a CLUE conference, each endpoint has a CLUE channel connecting it to an MCU. (In conferences with cascaded mixers [RFC4353], two MCUs will be connected by a CLUE channel.)

A.2.1. CLUE Channel Lifetime

The transport mechanism used for CLUE messages is DTLS/SCTP as specified in [I-D.tuexen-tsvwg-sctp-dtls-encaps] and [I-D.ietf-mmusic-sctp-sdp]. A CLUE channel consists of one SCTP stream in each direction over a DTLS/SCTP session. The mechanism for establishing the DTLS/SCTP session is described in Section 4.

The CLUE channel will usually be offered during the initial SIP INVITE, and remain connected for the duration of the CLUE/SIP session. However this need not be the case. The CLUE channel may be established mid-session after desire and capability for CLUE have been determined, and the CLUE channel may be dropped mid-call if the desire and/or capability to support it is lost.

There may be cases when it becomes necessary to "reset" the CLUE channel. This by be as a result of an error on the underlying SCTP association, a need to change the endpoint address of the SCTP association, loss of CLUE protocol state, or something else TBD.

The precise mechanisms used to determine when a reset is required, and how to accomplish it and return to a well defined state are TBS.

A.2.2. Channel Error Handling

We will need to specify behavior in the face of transport errors that are so severe that they can't be managed via CLUE messaging within the CLUE channel. Some errors of this sort are:

- Unable to establish the SCTP association after signaling it in SDP.
- CLUE channel setup rejected by peer.
- Error reported by transport while writing message to CLUE channel.
- Error reported by transport while reading message from CLUE channel.
- Timeout - overdue acknowledgement of a CLUE message. (Requirements for now soon a message must be responded to are TBD.)
- Application fault. CLUE protocol state lost.

The worst case is to drop the entire CLUE call. Another possibility is to fall back to legacy compatibility mode. Or perhaps a "reset" can be done on the protocol. E.g. this might be accomplished by sending a new O/A and establishing a replacement SCTP association. Or a new CLUE channel might be established within the existing SCTP association.

A.3. Message Framing

Message framing is provided by the SCTP transport protocol. Each CLUE message is carried in one SCTP message.

Authors' Addresses

Paul Kyzivat
Huawei

Email: pkyzivat@alum.mit.edu

Lennard Xiao
Huawei

Email: lennard.xiao@huawei.com

Christian Groves
Huawei

Email: Christian.Groves@nteczone.com

Robert Hansen
Cisco Systems

Email: rohanse2@cisco.com