Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 16, 2014

P. Kyzivat
L. Xiao
C. Groves
Huawei
R. Hansen
Cisco Systems
July 15, 2013

CLUE Signaling
draft-kyzivat-clue-signaling-04

Abstract

This document specifies how signaling is conducted in the course of CLUE sessions. This includes how SIP/SDP signaling is applied to CLUE sessions as well as defining a CLUE-specific signaling protocol that complements SIP/SDP and supports negotiation of CLUE application level data.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document specifies how signaling is conducted in the course of CLUE sessions. This includes how SIP/SDP signaling is applied to CLUE sessions as well as defining a CLUE-specific signaling protocol that complements SIP/SDP and supports negotiation of CLUE application level data.

[Yes, this is a dup of the abstract for now. Eventually it should say more.]

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 Channel: A reliable, bidirectional, transport mechanism used to convey CLUE messages. A CLUE channel consists of one SCTP stream in each direction over a DTLS/SCTP session.

3. CLUE-Specific Signaling Protocol

The CLUE Framework [I-D.ietf-clue-framework] mentions a CLUE-specific protocol for the exchange of ADVERTISEMENT and CONFIGURE messages, but gives little detail. The Data Model [I-D.presta-clue-data-model-schema] specifies a model and XML representation for CLUE-related data, but doesn't currently specify exactly what data belongs in each message, or how messages are sequenced. This document provides the detail missing from those documents.

3.1. Protocol Versioning, Options & Extensions

3.1.1. Versioning

There must be some provision for identifying incompatible protocol versions.

NOTE: We probably don't want to have incompatible versions. Typically changes will be introduced in a backward compatible way. But a time may come when this isn't possible, and we should be prepared for that. This is more likely to occur before an RFC is published. While it is probably unwise to deploy a product based on a draft, there will certainly be prototypes developed for testing, and those tests may lead to a need for incompatible change. So whatever the mechanism is, it should be applicable to changes that occur from draft to draft, as well as after an RFC has been published.

3.1.2. Options and/or Extensions

There must be some provision for dealing with optional-to-implement features in the specification, and/or for backward compatible extensions to the protocol. These are superficially different, but in practice they are more-or-less equivalent. To an implementation of the base protocol and some extensions, those extensions must be viewed as optional-to-implement features in peers.

One decision is whether extensions may be implemented mix-and-match, or whether there is a sequence of extensions, and one extension may only be supported if all the prior extensions have been supported.

3.1.3. Negotiation

Both version and options can be negotiated. Some mechanisms may work for both, while others are only appropriate for one or the other. Some possibilities:

o No negotiation at all. Instead, unrecognized syntax in certain "extension points" is to be ignored. If it is recognized, then a corresponding extension specification defines what to do.

o Negotiate via the SIP signaling.

o Negotiate as part of the O/A exchange that establishes the channel. (E.g. it is likely that individual channels of the SCTP association will be specified in SDP with a specific sub-protocol type. There could be a separate sub-protocol for each new version.)

- o Negotiate within the CLUE channel, via a special message exchange, before exchanging "normal" CLUE messages.

- o Declare versioning in every CLUE message. Define errors for unsupported versions and fallback to earlier versions.

3.1.4. Principles

- o CLUE SHOULD allow forwards and backwards compatibility through a version and extension mechanism. Forward compatibility allows a version of a protocol to communicate effectively with and interwork with future versions of the protocol. A version should not restrict the future protocol from providing extra capabilities.

- o Whenever possible backwards compatibility should be maintained. Backward compatibility rules will be defined to ensure that endpoints implementing future versions of CLUE will be able to send protocol messages of the previous versions which will be understood and fully processed by the remote endpoint.

- o Existing protocol elements should not be changed unless a protocol error needs to be corrected.

- o The semantics of existing elements and values should not be changed.

- o Established rules for formatting and encoding messages and elements should not be changed.

- o When information elements are found to be obsolete they can be marked as not used. However, the identifier for that information element will be marked as reserved. In that way it cannot be used in future versions.

3.2. Acknowledging Messages

The CLUE channel is reliable, so there is no need for acknowledgement to guarantee delivery. But there is still a need for application-to-application acknowledgement to report that the message has been received, parsed, and found to be of an acceptable format. One possibility is to introduce separate ACK and NAK messages. Another possibility is to add a confirmation element to each CLUE message, so that confirmation can be piggybacked on the basic messages. Some alternatives follow. [OTHER PROPOSALS WELCOME.]

3.2.1. Explicit Acknowledgment of Each Message

The characteristics of this approach are:

- There are separate request and response messages. (This is similar to SIP.)

- Every request message expects exactly one response message.

- Every request message carries a sequence number that identifies it.

- Each end of the connection assigns sequential sequence numbers to the requests it sends.

- Every response message carries the sequence number of the message to which it responds.

- Responses are to be sent promptly upon the receipt of a request. (Needs more detail.)

- Responses are either ACK or NAK. NAK responses also carry info describing the error.

- Each CONFIGURE message is to be understood in the context of the most recent ACKed ADVERTISEMENT message. A CONFIGURE message may be rejected if there is an outstanding ADVERTISEMENT for which no response has been received. (Or it may be accepted if the advertiser is able to do so meaningfully.)

3.2.2. Piggybacking ACK on Requests

The characteristics of this approach are:

- Every message carries a sequence number that identifies it.

- Each end of the connection assigns sequential sequence numbers to the messages it sends.

- Every message carries the sequence number of the last message received and found valid.

- If a message is received and found invalid, then a NAK message is sent that refers to it and indicates what is wrong with it.

- If a valid message is received and a new message needs to be sent in response, then the responding message implicitly acknowledges the prior message.

- If a valid message is received and there is no need to immediately send another message, then a NO-OP message is sent to acknowledge the received message. But a NO-OP message is never sent in response to a NO-OP message.

- Each CONFIGURE message is to be understood in the context of the most recent *acknowledged* ADVERTISEMENT message. A CONFIGURE message may be rejected if it doesn't acknowledge the most recently sent ADVERTISEMENT. (Or it may be accepted if the advertiser is able to do so meaningfully.)

The general format of every message is:

- sequence # of this message

- sequence # of most recently *received* and *valid* message

- message type (ADVERTISEMENT, CONFIG, NO-OP, NAK)

- body of the message, according to type

(The exact representation is TBD - by XML experts.)

There are loose ends to resolve here. In particular, how to acknowledge messages after NAKing one.

3.2.3. **Reporting Message Errors**

There needs to be a mechanism to report errors with other messages. The details of form, content, and usage still need to be specified, and need to be tuned to the details of the protocol. This could use distinct messages or be incorporated into the other messages. Errors this message must be able to report include:

Syntax error in message: The message has been disregarded due to a syntax error detected at the message level. The message does not conform to the productions of messages in [Protocol Document]. Used when the message cannot be parsed.

Sequencing Error: Sequence number has already been used, or is greater than the expected number. (Details of possible errors depend upon the specific sequence numbering mechanism.)

Version not supported: This indicates a lack of support for the protocol version indicated in the message header of the message. In the case of the version number being indicated in the message header, the message contents are disregarded.

Option not supported: This indicates a lack of support for the protocol option the used in the message. The message contents are disregarded.

Unknown capture identity: The received Configure message contains an unknown capture identity not previously declared by an Advertisement. The message contents are disregarded.

Invalid identity: The received message contains an invalid capture identity. For example a duplicated Capture scene identity or some other semantically incorrect usage. The message contents are disregarded.

Invalid value: The received message contains an invalid parameter value. The value is not according to the protocol definition in [protocol document] or according the extension documentation.

Missing element: The received message is missing an element. Certain parameters require multiple values, e.g. Point of capture requires X,Y,Z co-ordinates if one or more elements are missing this error code is used.

Conflicting parameters or values: The received message contains multiple values that may not be used together.

Invalid capture area: The received message defines a capture area that cannot be rendered in a sensible manner. For example the capture area does not define a quadrilateral region.

Invalid point of line of capture: The indicated co-ordinate for the point on line of capture is invalid. For example: does not lie between the point of capture and the area of capture or it is the same as the point of capture.

Invalid capture scene entry: The message contains an invalid capture scene entry. For example the capture scene entry contains more than one media type.

Invalid Simultaneous Set: The simultaneous set contained in the message is invalid. For example the simultaneous set refers to an undefined capture set or does not match the specified capture scene entries.

Invalid Configuration: The Configure message requests a configuration that the provider cannot support.

Invalid Advertisement reference: The Configure message refers to an invalid Advertisement. The message refers-to/depends-upon out-of-date ADVERTISEMENT message or provides an invalid reference.

3.3. Stand-alone messages or deltas?

Each message exchanged within a CLUE session could contain a complete description of the state it wishes to achieve. Or each message could describe just the changes that it wishes to make to the current state. Or the protocol could support both message forms. Which direction to pursue is TBD.

[Paul: while this does need to be decided, it is fundamentally just an optimization. IMO it does not have major impact on the other parts of this document, so I would prefer to continue deferring it until we are so far along with the remainder of the document that we can no longer defer it.]

3.4. Message Sequencing

There is a very basic introduction to this topic in section 4 (Overview) of the CLUE Framework [I-D.ietf-clue-framework]. After removing extraneous material it would look like:

```
           +-----------+                  +-----------+
           | Endpoint1 |                  | Endpoint2 |
           +----+------+                  +-----+-----+
                |                               |
                | ADVERTISEMENT 1               |
                |*****************************>|
                |                ADVERTISEMENT 2 |
                |<******************************|
                |                               |
                |                    CONFIGURE 1 |
                |<******************************|
                | CONFIGURE 2                   |
                |*****************************>|
                |                               |
```

But we need much more than this, to show multiple CONFIGUREs per ADVERTISEMENT, interleaving of ADVERTISEMENTs and CONFIGUREs in both directions, etc.

Message sequencing needs to be described at two levels:

o Basic sequencing of the CLUE messages themselves, without regard for the SIP/SDP signaling that may be going on at the same time.

This is useful to cover the basic concepts. That should be covered in this section. It provides context for understanding the more detailed treatment later.

This could include some simple state machines.

- In reality there is a complex dependency between CLUE signaling and SDP Offer/Answer exchanges carried in SIP signaling. So there is a need to describe the valid ways in which these two forms of signaling interact. That is covered in Section 5.

3.4.1. Signaling Changes in Provider State

Once a CLUE session has been established, ADVERTISEMENTs and CONFIGUREs exchanged, and media is flowing, a provider may experience a change in state that has an effect on what it wishes or is able to provide. In this case it may need to alter what it is sending and/or send a new ADVERTISEMENT. In some cases it will be necessary to alter what is being sent without first sending a new ADVERTISEMENT and waiting for a CONFIGURE conforming to it.

The following is a non-exhaustive list of situations and recommended actions:

- An advertised capture, that is not currently configured, is no longer available.

  To recover from this: Send a new ADVERTISEMENT that omits this capture.

- An advertised capture, that has been configured, is no longer available.

  To recover from this: (1) stop transmitting the configured encoding of this capture. (2) Send a new ADVERTISEMENT that omits this capture.

- The provider loses some resource and must reduce the frame rate, frame size, or resolution of a capture encoding.

  If the reduced values still fall within the advertised values for the capture then the change may be made without any further signaling.

  If the change must be outside the range of what was advertised, then the provider must cease transmitting the capture encoding. It then must send a new ADVERTISEMENT reflecting what it is now capable of delivering.

o New or changed scenes or scene geometry. For instance, the addition of a new scene containing presentation captures. Also, an MCU may make significant changes in what it advertises as new endpoints join a conference.

o [Add more]

3.4.2. Signaling Changes in Consumer State

If the Consumer for some reason looses the CLUE state information how does it ask for an Advertisement from the provider? There could be multiple possibilities. A error code approach? However error codes would typically be associated with a NACK so it may not be good for a Config message. Maybe send a message which means "send me a complete update". An alternative may be to release the connection or just do new signaling to establish a new CLUE session.

3.5. 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.)

3.5.1. 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.

3.5.2. 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:

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

3.6. CLUE Messages

CLUE messages are encoded in XML. The Data Model [I-D.presta-clue-data-model-schema] defines many/most of the elements from which CLUE messages are composed. This document specifies an XML schema that contains an element definition for each CLUE message, with much of the content of those elements being drawn from the Data Model.

3.6.1. ADVERTISEMENT Message

This message contains XML representations of captures, capture scenes, encoding groups, and simultaneous sets using the types defined for those in the Data Model [I-D.presta-clue-data-model-schema].

The XML definition for this is element <advertisement> in section Section 3.7

[[Currently this does not contain any representation of encodings. It assumes those will be defined in SDP.]]

3.6.2. CONFIGURE Message

This message optionally contains an XML representations of captureEncodings using the type defined in the Data Model [I-D.presta-clue-data-model-schema]. A configure message with no captureEncodings indicates that no captures are requested.

[[It currently also contains a reference to the request number of the advertisement it is based upon. Whether this should be present, or if it should implicitly reference the most recently acknowledged advertisement is TBD.]]

The XML definition for this is element <configure> in section Section 3.7

3.6.3. ACK Message

Need for, and details of, the ACK message are TBD.

The XML element <response> in section Section 3.7 could serve as the representation, either with no reason element, or a reason element with a special value.

3.6.4. NAK Message

Need for, and details of, the NACK message are TBD.

The XML element <response> in section Section 3.7 could serve to as the representation, with the reason element providing the details. Then the code value in the reason element should map to the errors in section Section 3.2.3.

3.7. Message Syntax

[[The following is a first cut at a schema for the actual messages in the clue protocol. It uses <encodingGroups> from the data model but not <encodings>. Rather, it assumes that encodings are described in SDP as m-lines with a text identifier, and that the identifier has the same value as the encodingIDs embedded in the <encodingGroups>. If we stick with this the data model should be adjusted to agree, but until then it should "work". The SDP encoding of the identifier is proposed to be 'a=label:ID', though 'a=mid:ID' is another candidate.]]

For now there only <advertisement> and <configure> are defined. More messages will be needed for acknowledgment.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema
   targetNamespace="urn:ietf:params:xml:ns:clue-message"
   xmlns:tns="urn:ietf:params:xml:ns:clue-message"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:dm="urn:ietf:params:xml:ns:clue-info"
   xmlns="urn:ietf:params:xml:ns:clue-message"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified">

<!-- Import data model schema -->
   <xs:import namespace="urn:ietf:params:xml:ns:clue-info"
              schemaLocation="clue-data-model-04-wip.xsd"/>

<!-- ELEMENT DEFINITIONS -->
<xs:element name="advertisement" type="advertisementMessageType"/>
<xs:element name="configure" type="configureMessageType"/>
<xs:element name="response" type="responseMessageType"/>

<!-- CLUE MESSAGE TYPE -->
<xs:complexType name="clueMessageType" abstract="true">
  <xs:sequence>
    <!-- mandatory fields -->
    <!-- TBS: version info -->
  </xs:sequence>
</xs:complexType>

<!-- CLUE REQUEST MESSAGE TYPE -->
<xs:complexType name="clueRequestMessageType" abstract="true">
 <xs:complexContent>
  <xs:extension base="clueMessageType">
   <xs:sequence>
     <!-- mandatory fields -->
     <xs:element name="requestNumber" type="xs:integer"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<!-- CLUE RESPONSE MESSAGE TYPE -->
<xs:complexType name="clueResponseMessageType">
 <xs:complexContent>
  <xs:extension base="clueMessageType">
   <xs:sequence>
     <!-- mandatory fields -->
```

```
        <xs:element name="requestNumber" type="xs:integer"/>
        <!-- optional fields -->
        <xs:element name="reason" type="reasonType" minOccurs="0"/>
        <xs:any namespace="##other"
            processContents="lax" minOccurs="0"/>
      </xs:sequence>
     </xs:extension>
    </xs:complexContent>
   </xs:complexType>

   <!-- CLUE ADVERTISEMENT MESSAGE TYPE -->
   <xs:complexType name="advertisementMessageType">
    <xs:complexContent>
     <xs:extension base="clueRequestMessageType">
      <xs:sequence>
        <!-- mandatory fields -->
        <xs:element name="mediaCaptures"
                    type="dm:mediaCapturesType"/>
        <xs:element name="encodingGroups"
                    type="dm:encodingGroupsType"/>
        <!-- The encodings are defined via identifiers in the SDP,
             referenced in encodingGroups -->
        <xs:element name="captureScenes"
                    type="dm:captureScenesType"/>
        <!-- optional fields -->
        <xs:element name="simultaneousSets"
                    type="dm:simultaneousSetsType" minOccurs="0"/>
        <xs:any namespace="##other"
                processContents="lax" minOccurs="0"/>
      </xs:sequence>
     </xs:extension>
    </xs:complexContent>
   </xs:complexType>

   <!-- CLUE CONFIGURE MESSAGE TYPE -->
   <xs:complexType name="configureMessageType">
    <xs:complexContent>
     <xs:extension base="clueRequestMessageType">
      <xs:sequence>
        <!-- mandatory fields -->
        <xs:element name="advertisementNumber" type="xs:integer"/>
        <!-- advertisementNumber is requestNumber
             of the advertisement-->
        <!-- optional fields -->
        <xs:element name="captureEncodings"
                    type="dm:captureEncodingsType" minOccurs="0"/>
        <xs:any namespace="##other"
                processContents="lax" minOccurs="0"/>
```

```
      </xs:sequence>
     </xs:extension>
    </xs:complexContent>
   </xs:complexType>

   <!-- REASON TYPE -->
   <xs:complexType name="reasonType">
     <xs:simpleContent>
       <xs:extension base="xs:string">
         <xs:attribute type="xs:short" name="code" use="required"/>
       </xs:extension>
     </xs:simpleContent>
   </xs:complexType>

   </xs:schema>
```

3.8. Message Framing

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

3.9. other

4. CLUE use of SDP O/A

4.1. Establishing the CLUE channel

The CLUE channel is usually offered in the first SIP O/A exchange between two parties in an intended CLUE session. The offer of the CLUE channel is the indicator that this SIP session is proposing to establish a CLUE session.

(However it is also acceptable to start with a non-CLUE SIP session and upgrade it to a CLUE session later.)

The mechanism for negotiating a DTLS/SCTP connection is specified in [I-D.ietf-mmusic-sctp-sdp]. We need to specify how to select the specific pair of SCTP streams that comprise the CLUE channel.

The presence of an active m-line for the CLUE channel in an SDP offer is an indication that the offer that the sender is CLUE-capable and hence can understand CLUE-specific syntax.

4.2. Representing CLUE Encodings in SDP

Many CLUE constructs have no good analog in SDP. Entities such as 'captures', which describe spatial and other properties of a capture

source such as a camera, are not tied directly to RTP streams, do not have negotiated properties and would prove a significant challenge to represent in SDP syntax (while also greatly increasing the size of the SDP).

However, two entities defined in the CLUE Framework [I-D.ietf-clue-framework] are a much closer fit for SDP: Encodings and Encoding Groups. Both describe RTP media properties and limitations, though unlike most SDP usage they describe the sender's capabilities, not the receiver's. Representing encodings in CLUE splits media limitations across two protocols, and risks duplicated and potentially contradictory information being sent in CLUE and SDP. As such we are exploring representing this information in SDP, with the decision to convey them in the CLUE messages only to be made if the SDP approach proves impractical.

This draft presents an attempt to describe CLUE encodings in SDP. As a decision has not yet been reached on how multiplexed RTP streams are to be expressed in SDP, at this stage the draft does so without multiplexing, using existing SDP attributes, with a seperate "m" line and hence port per unidirectional RTP stream. This is done with the understanding that when a decision is reached on new syntax for multiplexing RTP streams in SDP the CLUE SDP signaling will be modified to use it. Further, the framework document states that the multiplexing of streams by an implementation is optional, and in the case of a disaggregated system, with media streams going to different addresses, may not be possible.

With the current scheme of using existing syntax, an encoding is specified in SDP as a unicast "m" line, which MUST be marked as sendonly with the "a=sendonly" attribute or as inactive with the "a=inactive" attribute. The encoder capabilities of the stream are defined here using existing 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.

A receiver who wishes to receive a CLUE stream via this encoding requires a matching "a=recvonly" "m" line. 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. Representing CLUE Encoding Groups in SDP

As per the previous section, there would be advantages to conveying encoding group information in SDP. However, with current SDP syntax there is no way to express the encoding group limits defined in the Data Model [I-D.presta-clue-data-model-schema]. As such the current draft keeps encoding groups as part of the Advertisement message for the time being.

4.4. 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 channel. If a CLUE 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 negotiation has taken place as defined in this document. If no CLUE 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.

4.5. Ensuring interoperability with non-CLUE devices

A CLUE-capable device sending an initial SDP offer SHOULD include an "m" line for the CLUE channel, but SHOULD NOT include any other CLUE-controlled "m" lines. Once each side of the call is aware that the other side is CLUE-capable a new O/A exchange MAY be used to add CLUE-controlled "m" lines.

5. Interaction of CLUE 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.

```
                +----------+                       +-----------+
                |  Alice   |                       |    Bob    |
                |          |                       |           |
                +----+-----+                       +-----+-----+
                     |                                   |
```

```
                      |                                  |
                      | INVITE 1 (BASIC SDP+COMEDIA)     |
                      |--------------------------------->|
                      |                                  |
                      |                                  |
                      |     200 OK 2 (BASIC SDP+COMEDIA) |
                      |<---------------------------------|
                      |                                  |
                      |                                  |
                      | ACK 1                            |
                      |--------------------------------->|
                      |                                  |
                      |                                  |
                      |                                  |
                      |<########### MEDIA 1 ############>|
                      |   1 video A->B, 1 video B->A     |
                      |<################################>|
                      |                                  |
                      |                                  |
                      |                                  |
                      |<================================>|
                      |   CLUE CTRL CHANNEL ESTABLISHED  |
                      |<================================>|
                      |                                  |
                      |                                  |
                      | ADVERTISEMENT 1                  |
                      |*********************************>|
                      |                                  |
                      |                                  |
                      |                  ADVERTISEMENT 2 |
                      |<*********************************|
                      |                                  |
                      |                                  |
                      | INVITE 2 (+3 sendonly)           |
                      |--------------------------------->|
                      |                                  |
                      |                                  |
                      |                      CONFIGURE 1 |
                      |<*********************************|
                      |                                  |
                      |                                  |
                      |           200 OK 2 (+2 recvonly) |
                      |<---------------------------------|
                      |                                  |
                      |                                  |
                      | ACK 2                            |
                      |--------------------------------->|
                      |                                  |
```

```
                      |                                  |
                      |                                  |
                      |<########### MEDIA 2 ############>|
                      |   2 video A->B, 1 video B->A     |
                      |<################################>|
                      |                                  |
                      |                                  |
                      |           INVITE 3 (+2 sendonly) |
                      |<---------------------------------|
                      |                                  |
                      |                                  |
                      | CONFIGURE 3                      |
                      |*********************************>|
                      |                                  |
                      |                                  |
                      | 200 OK 3 (+2 recvonly)           |
                      |--------------------------------->|
                      |                                  |
                      |                                  |
                      |                                  |
                      |                            ACK 3 |
                      |<---------------------------------|
                      |                                  |
                      |                                  |
                      |                                  |
                      |<########### MEDIA 3 ############>|
                      |   2 video A->B, 2 video B->A     |
                      |<################################>|
                      |                                  |
                      |                                  |
                      |                                  |
                      v                                  v
```

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".

Alice receives Bob's message CONFIGURE 1 but 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 but 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 and non-CLUE-capable 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.

```
      +----------+                       +-----------+
      |   EP1    |                       |    EP2    |
      |          |                       |           |
      +----+-----+                       +-----+-----+
           |                                   |
           |                                   |
           | INVITE 1 (BASIC SDP+COMEDIA)      |
           |---------------------------------->|
           |                                   |
           |                                   |
           | 200 OK 1 (BASIC SDP+*NO*COMEDIA)  |
           |<----------------------------------|
           |                                   |
           |                                   |
           | ACK 1                             |
           |---------------------------------->|
           |                                   |
           |                                   |
           |                                   |
           |<########### MEDIA 1 ############>|
           |   1 video A->B, 1 video B->A      |
           |<################################>|
           |                                   |
           |                                   |
           |                                   |
           |                                   |
           v                                   v
```

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. Interoperation with Legacy SIP Devices

This may just describe how the degenerate form of the general mechanisms work for legacy devices. Or it may describe special case handling that we mandate as part of CLUE. Or it may just discuss non-normative things for implementors should consider.

11. CLUE over RTCWEB

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

12. 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.

o 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.

o 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]

13. What else?

14. 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.

15. IANA Considerations

TBD

16. Security Considerations

TBD

17. Change History

-03:

* 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.

18. References

18.1. 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-11 (work in progress), July 2013.

[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.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.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.

18.2. 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.

[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.

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