Implementing AV Streaming with UPnP Cloud and XMPP Jingle

Document Revision Date: December 31, 2015

CONTENTS

1 Introduction

This document describes the mechanisms available with UPnP Device Architecture 2.0 Annex C (UPnP Cloud or UCA), UPnP AV DCPs, UPnP Cloud DCPs and XMPP Jingle (http://xmpp.org/about-xmpp/technology-overview/jingle/) for supporting AV streaming. It covers both 2-Box and 3-Box scenarios and both HTTP and RTP sessions. It outlines the decision tree for setting up the connections and provides examples. It assumes that all device and control points are UCCDs and UCC-CPs respectively. The scenarios presented are for entities connected to the same *Cloud Account* but will work for other users that have been invited via Roster or MUC for sharing purposes.

1.1 Audience

The reader is assumed to be familiar with the UPnP Device Architecture 2.0 [UDA] and specifically Annex C, the UPnP AV DCPs, the UPnP Cloud DCPs, and XMPP.

To be more precise these are:

- The latest AV DCPs (v4), primarily [AV-ARCH], [MS4], [MR3], and [CDS4]; the rest can be found at http://upnp.org/specs/av/av4.
- The Cloud DCPs [PROXY], [CPROXY], and [CTS].

An overview of XMPP Jingle, primarily XEPs [XEP-0166], [XEP-0167], [XEP-0176], and [XEP-0177], is provided in this document along with its usage in UPnP Cloud AV Streaming.

The following Acronyms, Terms and Definitions also apply:

1.1.1 Acronyms

Table 1-1: Acronyms

Acronym	Description
FQDN	Fully Qualified Domain Name
ICE	Interactive Connectivity Establishment
NAT	Network Address Translation
STUN	Session Transversal Utilities for NAT
TURN	Transversal Using Relays around NAT
UCA	UPnP Cloud Annex
UCC	UPnP Cloud Capable
UCC-CP	UPnP Cloud Capable Control Point
UCCD	UPnP Cloud Capable Device
UCCD-M	UPnP Cloud Capable Device [Mobile]
UCS	UPnP Cloud Server
XMPP	Extensible Messaging and Presence Protocol

1.1.2 General Cloud Terms and Definitions

Cloud, in the context of UPnP, is the logical domain, not in the user's home(s), where their UCCDs and UCC-CPs connect, and their cloud based content resides.

Domain is a scoped access to a subset of all available cloud devices, services and users.

Account is a domain in the cloud consisting of a username and a credential. Also, the account can contain ancillary information such as address, telephone number, email information and possibly transactional information such as credit card information. An extended concept of an account is the combination of devices, services and users registered or interacting with the account.

User, in the context of UPnP cloud, is a uniquely identifiable participant that interacts with the UPnP cloud ecosystem. A user can be an account owner or participant and can be associated with multiple *Cloud Account*s.

Login is an identification process that allows a specific user to access their *Cloud Account* by confirming their username and credential. Login can also refer to the act of starting an active session with the *Cloud Account*. The login can be automated once initial login succeeds. Some minimal behavior equivalent to UPnP Public Role [DP] could be identified.

Owner is a user that has management rights over an account (or group) and the devices, services, and users allowed access within that account.

UPnP Cloud Capable (UCC) means that the device or control point (CP) is capable of interaction with UPnP Cloud Based Services.

1.1.3 Device and Control Point Terms and Definitions

UPnP Cloud Capable Control Point (UCC-CP) is a control point (CP) that can interact with UCCDs, UCODs, and UCOSs directly. Note that a UCC-CP is not a UCCD.

UPnP Cloud Capable Device (UCCD) is a non-virtual UPnP device that can interact directly with other UCCDs and UCC-CPs via the cloud, when in the home it can interact with legacy devices and control points over a home network.

Invited Device is a device from a *Cloud Account* that has received an invitation to be registered to a different *Cloud Account*.

1.1.4 Jingle Terms and Definitions

The following are from [XEP-0166] and have been localized for UCA usage.

Jingle is an XMPP extension that defines capabilities designed to enable one-to-one, peer-to-peer media sessions between XMPP entities, where the negotiation occurs over the XMPP signalling channel and the media is exchanged over a data channel that is usually a dedicated non-XMPP transport.

Application Format is the data format of the content type being established, which formally declares one purpose of the session (e.g., "audio" or "video"). This is the 'what' of the session (i.e., the bits to be transferred), such as the acceptable codec when establishing a video streaming session. In Jingle XML syntax the application format is the namespace of the <**description**/> element. The values related to the application format will be highly coupled with the [CDS4] *res@protocolinfo* property third field metadata.

Component is a numbered stream of data that needs to be transmitted between the endpoints for a given content type in the context of a given session. It is up to the transport to negotiate the details of each component. Depending on the content type, multiple components might be needed (e.g., one to transmit an RTP stream and another to transmit RTCP timing information). The values related to the application format will be highly coupled with the [CDS4] *res@protocolinfo* property first field metadata.

Content Type is a pair formed by the combination of one application format and one transport method.

Session is one or more content types negotiated between two entities. It is delimited in time by a session-initiate action and a session-terminate action. During the lifetime of a session, content types can be added or removed. A session consists of at least one content type at a time.

Transport Method is the method for establishing data stream(s) between entities. Possible transports might include ICE-UDP, ICE-TCP, Raw UDP, In-Band Bytestreams, SOCKS5 Bytestreams, etc. This is the 'how' of the session. In Jingle XML syntax this is the namespace of the <**transport**/> element. The transport method defines how to transfer bits from one host to another. Each transport method MUST specify whether it is "datagram" or "streaming" as described in the Transport Types section of this document.

1.2 Notation

For readability purposes this document uses fonts as in [UDA] to indicate specific protocol components or:

- Strings that are to be taken literally are enclosed in "`double quotes`".
- Words that are emphasized are printed in *italic*.
- Keywords that are defined by the UPnP Working Committee are printed using the *forum* character style.
- Keywords that are defined by the UPnP Device Architecture are printed using the **arch** character style.
- Keywords that are defined specific to the UPnP Device Architecture Annex C are printed using **UCA** character style.
- Keywords that are defined specific to XMPP are printed using **XMPP** character style.
- Keywords that are defined by a vendor are printed in *`Vendor`* style.
- A double colon delimiter, "::", signifies a hierarchical parent-child (parent::child) relationship between the two objects separated by the double colon. This delimiter is used in multiple contexts, for example: Service::Action(), Action()::Argument, parentProperty::childProperty.
- Example text will be in `green courier new` with a shaded background (not necessarily gray)[1].

Additionally, clarifying text is sometimes included. There are two general forms:

1) Extracts from relevant XMPP specification indicated by an XEP or RFC reference followed by text enclosed in a box (see below).

From XEP or RFC Extracted relevant text from XEP or RFC goes here.

2) Implementation warnings shown as a "caution sign symbol" with explanatory text (see below).

	Implementation caution goes here.

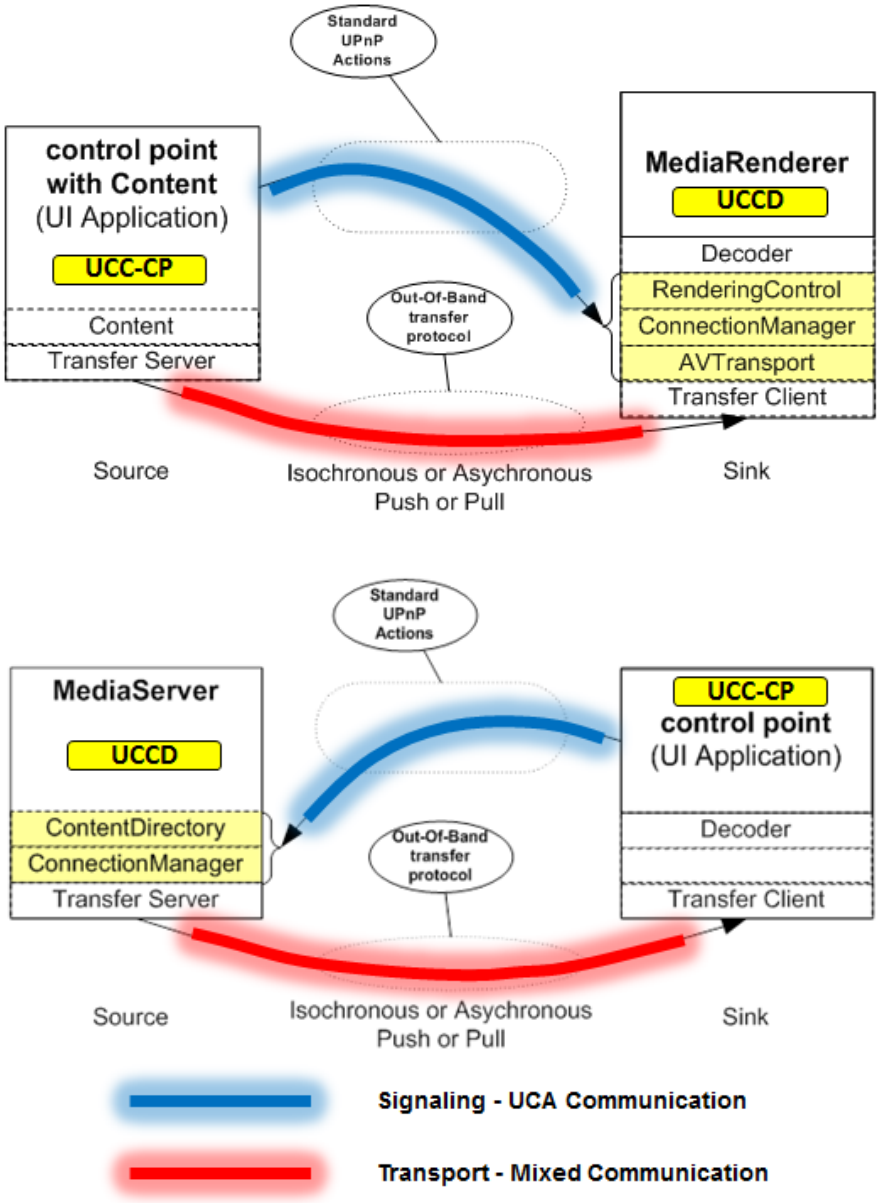
1.3 The AV Scenarios

The UPnP AV Architecture describes two general categories of AV streaming known as the 2-Box and 3-Box models. In the 2-Box models one of the endpoints is a device and one a control-point, in this case the control point will either contain the content and play it on a MediaRenderer device [MR3] or consume the content from a MediaServer device [MS4]. In the 3-Box model the content is on a MediaServer device and consumed on a MediaRenderer while the control point manages the setup, playback, and teardown of all interactions between the MediaServer and MediaRenderer with the only interaction between them being at the content transfer. For the scenarios in this document it is assumed that each device is also a

[1] Note: example text can contain whitespace and line feeds to improve readability.

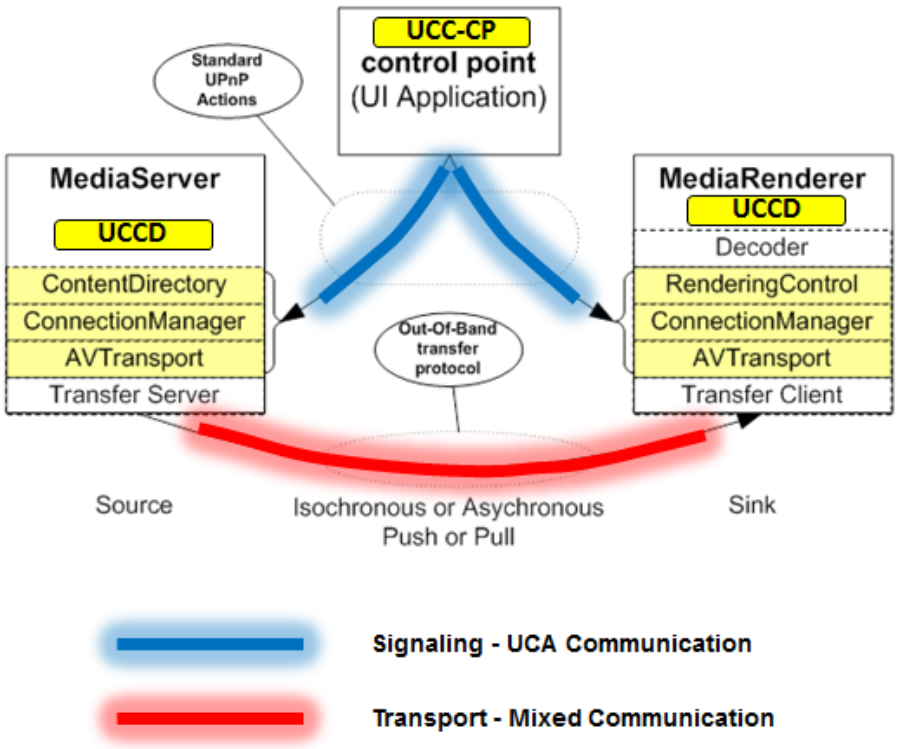
UCCD and each control point is a UCC-CP. Furthermore it is assumed that the UCCD supports the CloudTransport service [CTS]. It is possible that all UCCDs and Control Points in a 2-Box or 3-Box scenario are on the same local area network and although covered in this document the main use case is when the main connectivity is through the UCA interface. The 2-Box and 3-Box models are illustrated in Figure 1-1 and Figure 1-2 respectively.

Figure 1-1: 2-Box Models for UCC-CP content played on a MediaRenderer and MediaServer content played on a UCC-CP.



Note that the majority of interaction will be normal UPnP actions, including CloudTransport. over UCA (indicated as blue) and perhaps all depending on the actual connectivity, however, for efficiency purposes the transfer of the media content (indicated in red) could also take place using transport negotiated with Jingle (or HTTP if the content is reachable at a FQDN).

Figure 1-2: 3-Box Model where content is contained on a MediaServer UCCD and is played on a MediaRenderer UCCD under the control of a UCC-CP.



The detail of these interactions will be described in more detail later.

2 The Jingle XMPP Protocol Extension

This section introduces XMPP Jingle and is meant as a high level overview. To understand Jingle in detail it is suggested to read the primary Jingle XEPs [XEP-0166], [XEP-0167], [XEP-0176], and [XEP-0177], [XEP-0234], [XEP-0260], [XEP-0261] and [XEP-0293]. Jingle is an add-on capability usually supported on XMPP Client endpoints (UCC-CPs and UCCDs) but also requiring additional support from infrastructure components - STUN [RFC-5389] and TURN [RFC-5766], [RFC-6062] servers - for effective application. It can be discovered via XMPP Service Discovery [XEP-0030] <iq> stanzas exchanged between endpoints as a set of <feature> elements in the <query> response element. A typical <query> response element has the form shown below where

```
<query xmlns="http://jabber.org/protocol/disco#info">
```

the following elements indicate support for [XEP-0166], [XEP-0167] and both audio and video streams (basic Jingle).

```
    <feature var="urn:xmpp:jingle:1"/>
    <feature var="urn:xmpp:jingle:apps:rtp:1"/>
    <feature var="urn:xmpp:jingle:apps:rtp:audio"/>
    <feature var="urn:xmpp:jingle:apps:rtp:video"/>
```

The following elements indicate support for [XEP-0176] (ICE).

```
    <feature var="urn:ietf:rfc:3264"/>
    <feature var="urn:xmpp:jingle:transports:ice-udp:1"/>
```

The following element indicates support for [XEP-0177] (Raw UDP).

```
    <feature var="urn:xmpp:jingle:transports:raw-udp:1"/>
```

The following element indicates support for [XEP-0234] (TCP File Transfer over ICE).

```
    <feature var="urn:xmpp:jingle:apps:file-transfer:4"/>
```

The following element indicates support for [XEP-0293] (RTCP Feedback).

```
    <feature var="urn:xmpp:jingle:apps:rtp:rtcp-fb:0"/>
```

The following element indicates support for [XEP-0260] (SOCKS5 Transport over ICE).

```
    <feature var="urn:xmpp:jingle:transports:s5b:1"/>
```

The following element indicates support for [XEP-0261]

```
    <feature var="urn:xmpp:jingle:transports:ibb:1"/>

  </query>
```

Note that only support for the current version namespaces are indicated although **rtp:0** and **ice-udp:0** versions will likely also be supported.

Below is an example of a service discovery <**iq**> exchange between a UCC-CP and UCCD where all the above XEPs are supported.

UCC-CP jeffrey@upnp.org/urn:schemas-upnp-org:cloud-1-0:ControlPoint:1:ad93e8f5-634b-4123-80ca-225886a5c0e8
Sends:

```
<iq id="getjinglecapability1"
    from="jeffrey@mycloud.org/urn:schemas-upnp-org:
            cloud-1-0:ControlPoint:1:ad93e8f5-634b-4123-80ca-225886a5c0e8"
    to="jeffrey@mycloud.org/urn:schemas-upnp-org:device:
             MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-636a323e7950">
    type="get">
   <query xmlns="http://jabber.org/protocol/muc#info"/>
</iq>
```

UCC-CP jeffrey@upnp.org/urn:schemas-upnp-org:device:MediaServer:4:uuide70e9d0e-d9eb-4748-b163-636a323e7950
Sends:

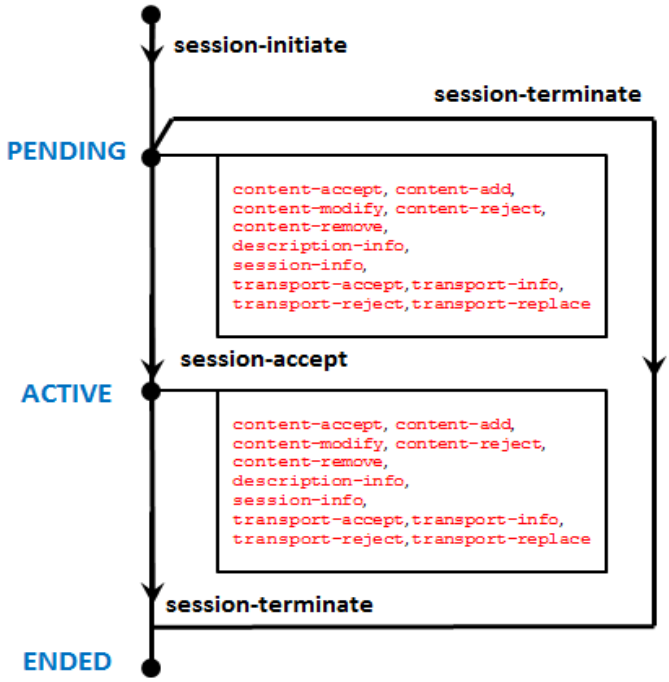
```
<iq id="getjinglecapability1"
    to="jeffrey@mycloud.org/urn:schemas-upnp-org:
            cloud-1-0:ControlPoint:1:ad93e8f5-634b-4123-80ca-225886a5c0e8"
    from="jeffrey@mycloud.org/urn:schemas-upnp-org:device:
            MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-636a323e7950">
    type="get">
   <query xmlns="http://jabber.org/protocol/muc#info">
     <feature var="urn:xmpp:jingle:1"/>
     <feature var="urn:xmpp:jingle:apps:rtp:1"/>
     <feature var="urn:xmpp:jingle:apps:rtp:audio"/>
     <feature var="urn:xmpp:jingle:apps:rtp:video"/>
     <feature var="urn:xmpp:jingle:apps:rtp:rtcp-fb:0"/>
     <feature var="urn:xmpp:jingle:apps:file-transfer:4"/>
     <feature var="urn:ietf:rfc:3264"/>
     <feature var="urn:xmpp:jingle:transports:ice-udp:1"/>
     <feature var="urn:xmpp:jingle:transports:raw-udp:1"/>
     <feature var="urn:xmpp:jingle:transports:s5b:1"/>
     <feature var="urn:xmpp:jingle:transports:ibb:1"/>
```

```
   </query>
</iq>
```

2.1 Jingle Session

The Jingle session has essentially 3 states - PENDING, ACTIVE, and ENDED. In the PENDING state the endpoints (an INITIATOR and a RESPONDER) are negotiating the parameters of the media exchange. In the ACTIVE state the parameters of the media exchange have been agreed and the media is being exchanged (note that the parameters can be modified during the ACTIVE state). And in the ENDED state either the INITIATOR or RESPONDER has terminated the session. Associated with each of these state transitions are a particular <iq>::<jingle> message with an action attribute of "session-initiate", "session-accept" and "session-terminate" for transition to PENDING, ACTIVE, and ENDED session states respectively.

Figure 2-1: State Transition Diagram for Jingle Sessions.



The initiating <iq> stanza would be in the form shown below. For UPnP Cloud typically the UCCD (or UCC-CP) containing the content to be streamed will possess the most detailed information about the content [2] and will be expected to provide the most accurate <description> element during session negotiation during the PENDNG state.

```
<iq
   from="Initiatorlocalpart@domainpart/UCAresourcepart">
   to="Responderlocalpart@domainpart/UCAresourcepart">
   id="vendor generated value"
   type="set"
   <jingle
      xmlns="urn:xmpp:jingle:1"/>
      action="session-initiate"
      initiator="Initiatorlocalpart@domainpart/UCAresourcepart"
      sid="vendor generated value">
      <content
        creator="Responderlocalpart@domainpart/UCAresourcepart"
```

[2] This might not be the case if a MediaServer hosting the content is using some of the advanced content description features of [CDS4].

```
         name="vendor defined value">
         <description xmlns="urn:xmpp:jingle:apps:rtp:1">
            detail in section 2.2
         </description>
         <transport xmlns="urn:xmpp:jingle:transport:ice-udp:1>
            detail in section 2.2
         </transport>
      </content>
   </jingle>
</iq>
```

The response to the session-initiate <iq> stanza is either an accept with a stanza of the form below,

```
<iq
   to="Initiatorlocalpart@domainpart/UCAresourcepart">
   from="Responderlocalpart@domainpart/UCAresourcepart">
   id="vendor generated value received from initiator"
   type="result"
</iq>
```

or an <iq> stanza of type "error". Typical error reasons are:

- The responder does not communicate with unknown entities.
- The responder does not support Jingle.
- The responder wishes to redirect the request to another address.
- The responder does not have sufficient resources to participate in a session.
- The initiation request was malformed.

More detail can be found in section 6.3.2 of [XEP-0166]. The response is typically sent immediately so that the PENDING state can be entered and session negotiation started.

2.2 Jingle <Description> and <Transport>

In the PENDING phase the INITATOR and RESPONDER will agree on Application Formats Content Types, and one or more Components primarily using the <description> and <transport> elements. Although Jingle is capable of supporting 2-way real-time communication with voice and video exchanged in both directions, for the UPnP AV streaming scenarios there is typically only a single content stream in one direction and an RTCP feedback stream in the reverse direction. The Content Type will be either "audio" or "video" corresponding to the UPnP *item* class of *object.item.audioItem* and *object.item.videoItem* respectively; or *componentClass* of "*audio*" or "*video*" if the *upnp:componentInfo* property is supported (see [CDS4] for details). In this document the single direction, single media component stream constraint is assumed.

The <description> element version 1 is defined in [XEP-0167]. It has the general form shown below. It can contain 0 or more <payload> elements, 0 or 1 <bandwidth> elements, and 0 or 1 <encryption> elements. As previously mentioned this element is used primarily to negotiate the Application Format.

```
   <description
       xmlns="urn:xmpp:jingle:apps:rtp:1"
       media="audio|video">
      <payload-type
         id="RTP Payload identifier according to IANA such as 10
             for 2-channel LPCM audio or 33 for MPEG2 Transport Stream AV"
         name="registered name such as MP2T for MPEG2 Transport Stream"
         clockrate="clock rate such as 90000 for MP2T"
         channels="such as 1 for mono or 2 for stereo"
         maxptime="Max packet time recommended by RFC 4566"
         ptime="Max packet time recommended by RFC 4566">
         <parameter
            name="name of parameter associated with the media type such as
                  width and height for video"
            value="string value representing parameter associated
                   with name value such as 1080 for video width"/>
      </payload-type>
```

```
      <bandwidth
         type="typically an SDP bwtype parameter"
         value="allowable of preferred bandwidth for the session
                in kilobits per second, see section 6.2 of RFC 3550"/>
      <encryption>
         <crypto
            crypto-suite="maps to SDP crypto-suite parameter such as:
                          AES_CM_128_HMAC_SHA1_80"
            key-params="maps to SDP key-params parameter such as:
                        inline:PS1uQCVeeCFCanVmcjkpPywj
                        NWhcYD0mXXtxaVBR|2^20|1:32"
            session-params="maps to SDP session-params parameter such as:
                            KDR=1;UNENCRYPTED_SRTCP"
            tag="maps to SDP tag parameter such as: 1"/>
      </encryption>
   </description>
```

Refer to sections 6 and 7 of [XEP-0167] for further details.

The <**transport**> element version **ice-upd:1** and **raw-upd:1** are defined in [XEP-0176] and [XEP-0177] respectively. It has the general form shown below. It can contain 0 or more <**candidate**> elements, 0 or 1 <**remote candidate**> elements. Note that the **raw-udp:1** support is a simplified version of **ice-upd:1** that lacks basic connectivity checks and NAT traversal capabilities or basically a "do you feel lucky?" approach. It is primarily intended for the circumstance when the sending entity is a relay or gateway. It is expected that the UPnP Cloud AV streaming scenarios with *protocolInfo* first field of "*rtsp-rtp-udp*"[3] will be of the type most suitable for **ice-udp:1** and a focus of this document. As previously mentioned this element is used primarily to negotiate the Transport Method.

```
   <transport
      xmlns="urn:xmpp:jingle:transports:ice-udp:1|
             urn:xmpp:jingle:transports:ice-udp:1"
      pwd="RESPONDER or INITIATOR generated password, see [RFC-5245]"
      ufrag="RESPONDER or INITIATOR generated password, see [RFC-5245]">
     <candidate
        component="A Component ID as defined in [RFC-5245] such as 1"
        foundation="A Foundation as defined in [RFC-5245] such as 1"
        generation="An index, such as 0, defined in ICE Restarts
                    section of [XEP-0176]"
        id="vendor generated unique value for the candidate"
        ip="IP address for the candidate transport mechanism such as 10.0.1.1"
        network="An network index for diagnostics such as 1"
        port="The port at the candidate IP address such as 8998"
        priority="A priority as defined in [RFC-5245] such as 2130706431"
        rel-addr="A related address as defined in [RFC-5245] such as 10.0.1.1"
        rel-port="A related port as defined in [RFC-5245] such as 8998"
        protocol="udp"
        type="host|prflx|relay|srrfx"/> Candidate Type as defined
                                        in [RFC-5245] corresponding to
                                        HOST, Peer Reflexive, Relayed,
                                        and Server Reflexive candidates.
      </transport>
```

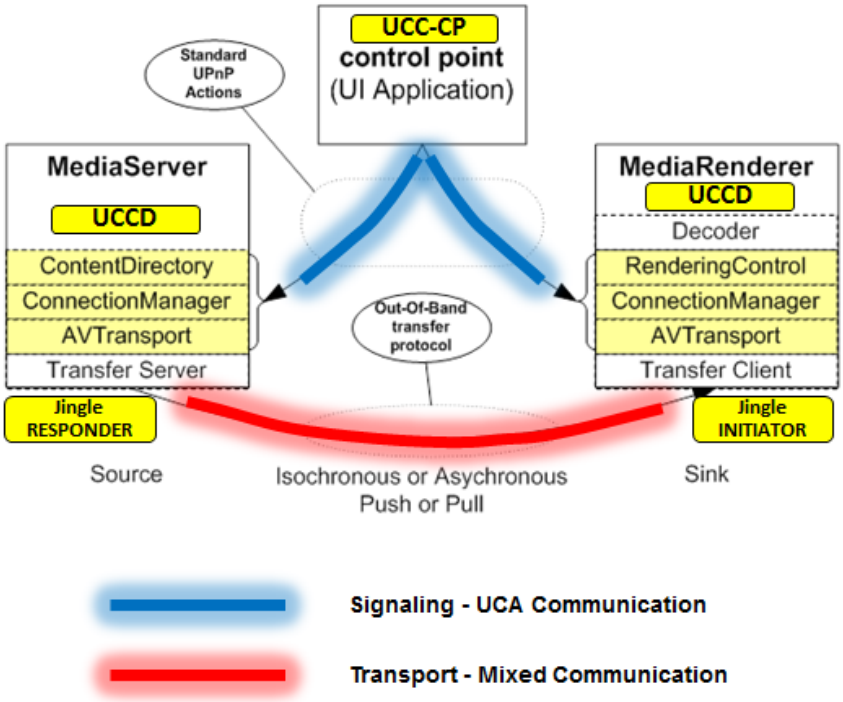
Refer to [XEP-0176] for further details.

2.3 Negotiating the Jingle Setup

If the UPnP Cloud endpoints have determined that Jingle is to be used to set up the AV Streaming session then the session can proceed as described below. Note that in general the UCCD (MediaRenderer) or UCC-CP that is to render the content will be the INITIATOR and the UCCD (MediaServer) or UCC-CP that serves the content will be the RESPONDER. For example the 3-Box model shown in Figure 1-2 now becomes the configuration in Figure 2-2**.**

[3] It could be useful to use a new *protocolInfo* 1st field of value "*jingle-rtp-udp*" when Jingle is supported.

Figure 2-2: 3-Box Model with Jingle INITIATOR and RESPONDER.



Thus when the Jingle INITIATOR sends the Jingle <**iq**> **session-initiate** stanza it includes either the *item@id* or the URI of the content item in the <**content**> element **name** attribute and sets the **creator** attribute to a value of "**responder**".

For example if the following content item is available from a MediaServer

```
<item id="InternetStream1" restricted="0">
   <dc:title>Some Stream</dc:title>
   <upnp:class>
      object.item.videoItem
   </upnp:class>
   <res protocolInfo="rtsp-rtp-udp:*:MP4:*">
      rtsp://192.168.1.23/stream1.mp4
   </res>
   <res protocolInfo="http-get:*:video/mp4:*">
      http://192.168.1.23/stream1.mp4
   </res>
</item>
```

or the URI has been set on a MediaRenderer using the *SetAVTransportURI()* action then the <**iq**> stanza would look like the form below when the content is indicated by a URI:

```
<iq
   from="Initiatorlocalpart@domainpart/UCAresourcepart">
   to="Responderlocalpart@domainpart/MediaServerUCCDresourcepart">
   id="vendor generated value"
   type="set"
   <jingle
      xmlns="urn:xmpp:jingle:1"/>
      action="session-initiate"
      initiator="Initiatorlocalpart@domainpart/UCAresourcepart"
      sid="vendor generated value">
      <content
         creator="Responderlocalpart@domainpart/MediaServerUCCDresourcepart"
         name="rtsp://192.198.1.23/stream.mp4">
```

```
         <description xmlns="urn:xmpp:jingle:apps:rtp:1">
            some best effort minimum description allowing RESPONDER to
            offer more detailed options
         </description>
         <transport xmlns="urn:xmpp:jingle:transport:ice-udp:1>
            Some transport options to be described in section 2.4
         </transport>
      </content>
   </jingle>
</iq>
```

and the form below when the content is indicated by a content item.

```
<iq
   from="Initiatorlocalpart@domainpart/UCAresourcepart">
   to="Responderlocalpart@domainpart/MediaServerUCCDresourcepart">
   id="vendor generated value"
   type="set"
   <jingle
      xmlns="urn:xmpp:jingle:1"/>
      action="session-initiate"
      initiator="Initiatorlocalpart@domainpart/UCAresourcepart"
      sid="vendor generated value">
      <content
         creator="Responderlocalpart@domainpart/MediaServerUCCDresourcepart"
         name="item@id::InternetStream1">4
         <description xmlns="urn:xmpp:jingle:apps:rtp:1">
            some best effort minimum description allowing RESPONDER to
            offer more detailed options
         </description>
         <transport xmlns="urn:xmpp:jingle:transport:ice-udp:1>
            Some transport options to be described in section 2.4
         </transport>
      </content>
   </jingle>
</iq>
```

Once the **session-initiate** <**iq**> is received by the RESPONDER then it sends an enhanced offer with additional <**description**>::<**payload-type**> elements using a **content-add** response. If the initiator is satisfied with the offer then it can transition the Jingle session to the ACTIVE state by sending an <**iq**> stanza with a **session-accept** type with an enumeration of the acceptable Content Type. See detailed example in 4. Recall that in the streaming case there js typically only one component to negotiate and that is the content stream, however, RTCP feedback is typically including in the RTP but it is negotiated as part of the same component. Negotiation of RTCP feedback is described in [XEP-0293].

2.4 The Media Offer <description> and Acceptance

In most UPnP AV Streaming scenarios the media Application Format matching is basically mature (see http://dlna.org). Therefore only small refinements will likely be needed to the content offer such as a bandwidth refinement. For example the *res@protocolInfo* property 4th field of content *item* with:

```
<item id="DLNAStream1" restricted="0">
   <dc:title>Some Stream</dc:title>
   <upnp:class>
      object.item.videoItem
   </upnp:class>
   <res protocolInfo="rtsp-rtp-udp:*:video/mpeg:DLNA.ORG_PN=AVC_TS_NA_ISO">
      http://192.168.0.20:58849/media/vid0001.mpeg
   </res>
</item>
```

4 Note that the suggested format is string "item@id::" concatenated with the actual string value of the *item id*.

indicates AVC video wrapped in MPEG-2 transport stream, constrained by SCTE standards, with either AC-3, Enhanced AC-3, MPEG-4 HE-AAC v2 or MPEG-1 Layer II audio, without a timestamp field. Thus the MediaRenderer would have a high probability of knowing how to construct the offer and knowing that it could render it upon delivery. Therefore, the offered <**description**> can be very tailored[5] to the particular component.

2.5 The ICE/STUN/TURN Offer <transport> and Acceptance

There is an underlying assumption that each Jingle Client has been configured with knowledge of a STUN and TURN server (the best practice is to have the servers co-located). Figure 2-3 illustrates the general architecture of UPnP Cloud with Jingle ICE-STUN-TURN support. Using ICE the Jingle end-points first gather candidate IP addresses for the session from different sources These candidates are (from [RFC-5245]):

- Host Candidate: A candidate obtained by binding to a specific port from an IP address on the host. This includes IP addresses on physical interfaces and logical ones, such as ones obtained through Virtual Private Networks and Realm Specific IP.

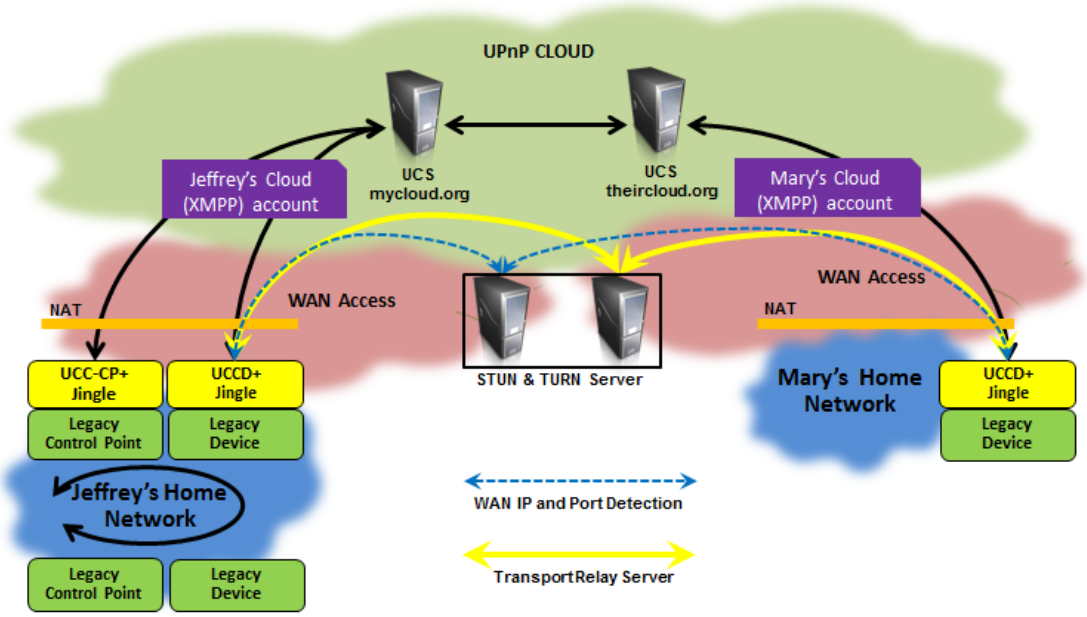
- Server Reflexive Candidate: A candidate, obtained from a STUN server, whose IP address and port are a binding allocated by a NAT for an agent when it sent a packet through the NAT to a server. Server reflexive candidates can be learned by STUN servers using the Binding request, or TURN servers, which provides both a relayed and server reflexive candidate.

- Peer Reflexive Candidate: A candidate whose IP address and port are a binding allocated by a NAT for an agent when it sent a STUN Binding request through the NAT to its peer.

- Relayed Candidate: A candidate obtained by sending a TURN Allocate request from a host candidate to a TURN server. The relayed candidate is resident on the TURN server, and the TURN server relays packets back towards the agent.

The candidates are then shared using the <**transport**> element.

Figure 2-3: UPnP Cloud with ICE-STUN-TURN.



[5] Note that this is somewhat restrictive in respect to typical SDP where the full capabilities are exchanged.

2.5.1 UPnP Remote Access and STUN/STUN

The following is an updated excerpt (through Table 2-1) from the UPnP Remote Access Architecture 2 [RAARCH] which provides specific information regarding connecting UPnP devices and control points through NATs. Network Address Translation (NAT) has been deployed by some Service Providers to deal with the IPv4 address exhaustion issue. NAT is deployed in routers and helps to reduce the IPv4 address usage by supporting multiple devices behind a single public IP address. It allocates private IP addresses [RFC-1918][6] to these devices and manages the dynamic translation/mapping between the internal and external IP addresses and ports. These dynamic address translations create a problem for remotely connecting devices that try to establish a connection to a remote endpoint because the “external” IP addresses and ports to the remote access endpoint do not stay static.

There are a variety of NAT implementations available in the market today. Description of the NAT behaviors are outside the scope of [RAARCH] but a detailed description of NAT behavior is available in [RFC-4787][7]. The following NAT behaviors are addressed in [RAARCH]:

- Endpoint Independent Mapping (also referred to as Full Cone NAT)
- Address Dependent Mapping (also referred to as Restricted Cone)
- Address and Port Dependent Mapping (also referred to as Port Restricted NAT)
- (Unique) Address and Port Dependent Mapping (also referred to as Symmetric NAT)

The following techniques are considered:

- Dynamic Domain Name System (DNS) update [RFC-2136][4]
- Domain Name System Resource Record for Location of Services (DNS SRV) [RFC-2782][4]
- Session Traversal Utilities for NAT (STUN) [RFC-5389]
- Traversal using relays around NAT (TURN) [RFC-5766]
- Connection Reversal mechanism for establishing a P2P connection [RFC-5128][4]
- Hole Punching mechanism for establishing a P2P connection [P2P Com] [4]

Table 2-1 below summarizes the use of the above techniques and combinations of these techniques to help establish a connection between a client and a server in the presence of the various NAT permutations.

[6] Refer to for [RAARCH] active links.

[7] Refer to for [RAARCH] active links.

Table 2-1: Summary of NAT Traversal Permutations.

Client / Server	1. Routable IP Address	2. Full Cone	3. Restricted Cone	4. Port Restricted	5. Symmetric NAT
1. Routable IP Address	DDNS-SRV	Same as 1.1 (+)	Same as 1.1	Same as 1.1	Same as 1.1
2. Full Cone	STUN, DDNS-SRV	Same as 2.1	Same as 2.1	Same as 2.1	Same as 2.1
3. Restricted Cone	STUN/TURN, DDNS-SRV, Connection Reversal	Same as 3.1	STUN/TURN, DDNS-SRV, Hole Punching	Same as 3.3	Same as 3.3
4. Port Restricted	STUN/TURN, DDNS-SRV, Connection Reversal	Same as 4.1	Same as 4.1 + Hole Punching	STUN/TURN (with relay), DDNS-SRV (*)	Same as 4.4 (*)
5. Symmetric NAT	STUN/TURN, DDNS-SRV, Connection Reversal	Same as 5.1	Same as 5.1 + Hole Punching	STUN/TURN (with relay), DDNS-SRV (*)	Same as 5.4 (*)

(+) "same as 1.1" means the solution is the same as row 1 column1
(*) indicates the data is relayed through the TURN server.

2.5.2 UPnP Cloud Specific Jingle ICE-UDP Setup

Elaborating on the example in [XEP-0176], the INITIATOR (a MediaRenderer) needs to stream content from a RESPONDER (a MediaServer). They send transport candidates, 2 for the INITIATOR and 1 for the RESPONDER, as shown below in the <**transport**> element fragments. For the INITIATOR the highest priority candidate is its local IP address `10.0.1.1` and port `8998`.

INITIATOR

```
      <transport xmlns="urn:xmpp:jingle:transports:ice-udp:1"
                 pwd="asd88fgpdd777uzjYhagZg"
                 ufrag="8hhy">
        <candidate component="1"
                   foundation="1"
                   generation="0"
                   id="el0747fg11"
                   ip="10.0.1.1"
                   network="1"
                   port="8998"
                   priority="2130706431"
                   protocol="udp"
                   type="host"/>
        <candidate component="1"
                   foundation="2"
                   generation="0"
                   id="y3s2b30v3r"
                   ip="192.0.2.3"
                   network="1"
                   port="45664"
                   priority="1694498815"
                   protocol="udp"
                   rel-addr="10.0.1.1"
                   rel-port="8998"
                   type="srflx"/>
      </transport>
```

The RESPONDER candidates are shown below. In this case, it only has one candidate address:port combination.

RESPONDER

```
      <transport xmlns="urn:xmpp:jingle:transports:ice-udp:1"
                 pwd="YH75Fviy6338Vbrhrlp8Yh"
                 ufrag="9uB6">
        <candidate component="1"
                   foundation="1"
                   generation="0"
                   id="or2ii2syr1"
                   ip="192.0.2.1"
                   network="0"
                   port="3478"
                   priority="2130706431"
                   protocol="udp"
                   type="host"/>
      </transport>
```

Once candidates are received the endpoints begin connectivity checks on candidate pairs. A partial sequence is shown in Figure 2-4 (but does not shown details of the connectivity check in the INITIATOR to RESPONDER direction).

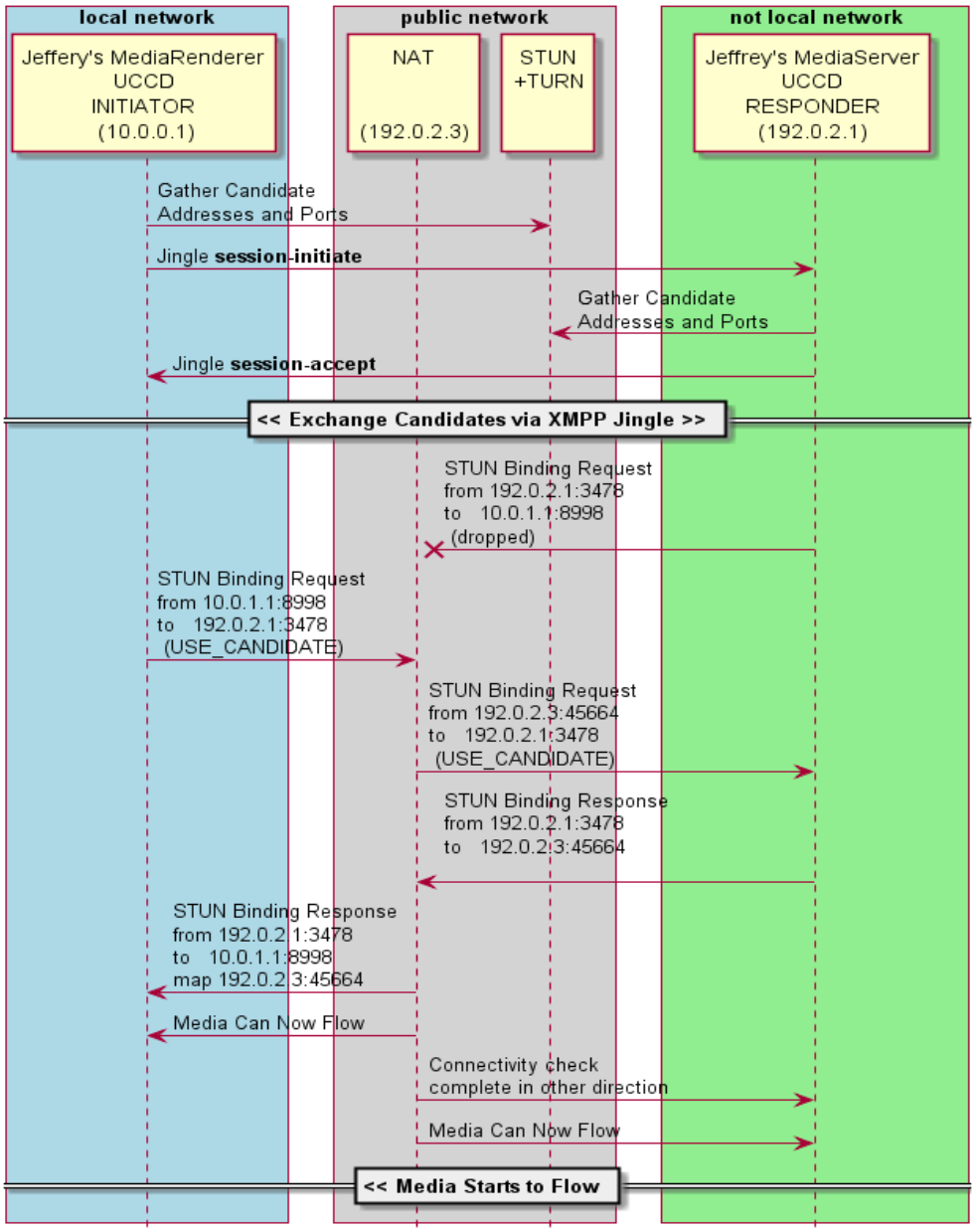
Once the two-way connectivity check has been completed, and confirmed by the INITIATOR with an <**iq**>::<**jingle**> stanze of **action** type **transport-info** as shown below, then the media content is ready to send.

UCC-CP to UCCD MediaServer

```
<iq
   from="jeffrey@mycloud.org/urn:schemas-upnp-org:device:
           MediaRenderer:3:0ee79d0e-e8f5-80ca-4123-225886a58850
   to="jeffrey@mycloud.org/urn:schemas-upnp-org:device:
           MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-636a323e7950">
    id="pd81b49s"
    type="set">
 <jingle xmlns="urn:xmpp:jingle:1"
         action="transport-info"
         initiator="jeffrey@mycloud.org/urn:schemas-upnp-org:device:
           MediaRenderer:3:uuid:0ee79d0e-e8f5-80ca-4123-225886a58850"
         sid="a73sjjvkla37jfea">
   <content creator="initiator" name="this-is-the-audio-content">
     <transport xmlns="urn:xmpp:jingle:transports:ice-udp:1"
                pwd="asd88fgpdd777uzjYhagZg"
                ufrag="8hhy">
       <remote-candidate component="1"
                         ip="10.0.1.2"
                         port="9001"/>
     </transport>
   </content>
 </jingle>
</iq>
```

Note that additional transport negotiation can take place during the media session, for example, if some adaptation is needed on the streams, however the details are out of scope for this document. Refer to [XEP-0166], [XEP-0167], [XEP-0176] for additional details on the Jingle session for re-negotiation as well as security considerations. Also, for a further overview on ICE see (http://www.internetsociety.org/articles/interactive-connectivity-establishment).

Figure 2-4: ICE Transport Negotiation Example



3 UCA Media Setup Decision Logic

This section describes a general AV streaming decision logic for setting up the best AV transport for a desired streaming experience. The main criteria for the decision logic in prioritized order are:

1. avoid sending content bitstream data through the UCS,
2. avoid packet loss,
3. avoid sending content bitstream data through the TURN relay server,
4. avoid sending content bitstream data in the clear.

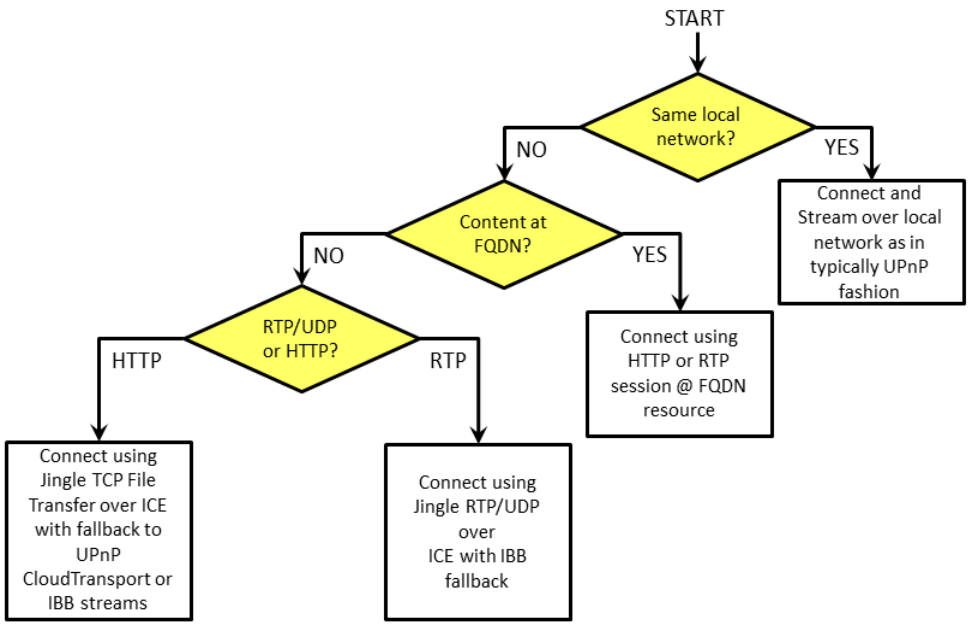
3.1 Top Level Strategy

With these criteria in mind then the top-level decision logic is:

1. determine if the connecting endpoints (UCCDs and UCC-CPs) are in the same local network, if yes, then use the UDA interface for content transfer, if not then,
2. determine if the content is at a reachable URI, i.e. a FQDN, if yes stream the content from the location using the suggested protocol, if not then,
3. determine if the content and transport is compatible with HTTP, if yes then attempt to set up a Jingle TCP File Transport session [XEP-0234], if not supported then, use the UPnP CloudTransport service (if the UCA endpoints are compatible) or XEP In-Band-ByteStreams [XEP-0261] file transfer.
4. or, determine if the content and transport is compatible with RTP/UDP, if yes then attempt to set up a Jingle RTP session [XEP-0176] with RTCP feedback [XEP-0293] if needed, if not supported then use XEP In-Band-ByteStreams [XEP-0261] file transfer.

This top-level decision logic is illustrated in Figure 3-1 and further refined in the remainder of this section.

Figure 3-1: Top Level Decision Logic.



2-Box and 3-Box Connectivity Considerations

For the 2-Box model the UCC-CP should have direct knowledge of whether the UCCD it will create an AV streaming session with is in the same local network, therefore the first decision point will be straight forward. This is not the case for the 3-Box model since 1) the UCC-CP could be on a completely separate local network (or no local network at all if it is strictly a Cloud based application) and 2) UCCDs (devices) are not typically aware of other UCCDs (devices). In this case the UCCD MediaRenderer or MediaServer could try the direct connection method first and wait for failure to trigger the Jingle session or they could initiate an ICE Jingle session and see if the local-to-local candidate pair connects and if so then terminate the ICE session and use the direct method.

Note that if the first method is used, some implementations could have considerable delay before failure resulting in a less than satisfactory user experience. Note also that such techniques as pre-fetching can be used in many cases to hide some of this delay from users.

As previously stated it is assumed that all UCCD MediaServers and MediaRenderers will support the CloudTransport service either directly or via a CloudProxy. However, this leaves some gaps in playback coverage - mainly:

- 3-Box case where the UCCD MediaRenderer does not have a CloudTransport control point,
- 2-Box case where the content is on a UCC-CP which does not have a CloudTransport service.

In the first case (3-Box) this is easily solved by either implementing an embedded CloudTransport control point on the UCCD MediaRenderer or falling back directly to IBB (bandwidth wise these are practically equivalent). In the latter case direct or Jingle assisted direct can be used. The 2-Box and 3-Box considerations are summarized in Table 3-1.

Table 3-1: Summary of 2-Box and 3-Box Connectivity Considerations.

2-Box 3-Box	**HTTP Content of UCC-CP**	**RTP Content on UCC-CP**	**HTTP Content on UCCD Media Server**	**RTP Content on UCCD Media Server**
HTTP Playback on MediaRender	direct or Jingle assisted direct, Jingle ICE-TCP, IBB	X[8]	direct or Jingle assisted direct, Jingle ICE-TCP, CloudTransport[9], IBB	X
RTP Playback on MediaRender	X	direct or Jingle direct, Jingle ICE-RTP-UDP, IBB	N/A	direct or Jingle assisted direct, Jingle ICE-RTP, CloudTransport, IBB
HTTP Playback on UCC-CP	X	X	direct, Jingle ICE-TCP, CloudTransport, IBB	X
RTP Playback on UCC-CP	X	X	X	direct or Jingle assisted direct, Jingle ICE-RTP-UDP, IBB

3.1.1 Session Control

Typical session control includes the ability to Play, Pause, and Stop media and can include other features such as Fast-Forward, Reverse, and advance features referred to as trick modes, such as, 2X or ½X forward and reverse playspeeds. For UPnP devices the first set is usually signalled using the AVTransport Service actions such as *Play()*, *Pause()* and *Stop()* while the latter are signalled using HTTP or RTSP constructs. Additionally, the AVTransport service can be either on the sender or receiving endpoint, or even both. Table 3-2 and Table 3-3 summarize some of the options and permutaions for the various playback states at a high level. Details are left to the implemented scenarios and interoperability choices such as using DLNA MediaServer (DMS) UCCD and MediaRender (DMR) UCCD devices. Note that if only simple support is needed, that is, Play, Pause, and Stop then a more direct mapping to HTTP and RTP methods can be made. A more detailed example is included in section 4.

[8] X indicates not applicable either due to capabilities or content mismatch.

[9] requires co-located CloudTransport control point on MediaRenderer.

Table 3-2: Summary of Session Controls AVTransport Service on Media Serving Endpoint.

2-Box 3-Box	HTTP Content of UCC-CP	RTP Content on UCC-CP[10]	HTTP Content on UCCD Media Server	RTP Content on UCCD Media Server
HTTP Playback on MediaRender	Modify HTTP bitstream directly on Server side	X[11]	Use AVTransport *PLAY()*, etc; buffer media on receiver for advanced options	X
RTP Playback on MediaRender	X	Modify RTP bitstream directly on Server side	X	Use AVTransport *PLAY()*, ect to modify RTP; buffer media on receiver for advanced options
HTTP Playback on UCC-CP	Modify HTTP bitstream directly on Server side	X	Use AVTransport *PLAY()*; buffer media on receiver for advanced options	X
RTP Playback on UCC-CP	X	Modify RTP bitstream directly on Server side	X	Use AVTransport *PLAY()*, etc to modify RTP; buffer media on receiver for advanced options

Table 3-3: Summary of Session Controls AVTransport Service on Media Rendering Endpoint.

2-Box 3-Box	HTTP Content of UCC-CP	RTP Content on UCC-CP	HTTP Content on UCCD Media Server	RTP Content on UCCD Media Server
HTTP Playback on MediaRender	Modify HTTP bitstream directly on Server side	X	Signal HTTP Throttle using AVTransport; buffer media on receiver for advanced options or signal via HTTP Header	X
RTP Playback on MediaRender	X	Modify RTP bitstream directly on Server side	X	Signal RTP modify using AVTransport; buffer media on receiver for advanced options
HTTP Playback on UCC-CP	Modify HTTP bitstream directly on Server side	X	Use AVTransport *PLAY()*; buffer media on receiver for advanced options	X
RTP Playback on UCC-CP	X	Modify RTP bitstream directly on Server side	X	Signal RTP modify using AVTranport; buffer media on receiver for advanced options

[10] RTP keeps alives will be needed when sender is in paused state or transitioning to next playstate.

[11] X indicates not applicable either due to capabilities or content mismatch.

3.1.2 Jingle Session Teardown and Clean-up Considerations

To end a Jingle session, whether in the PENDING or ACTIVE state, either the INITIATOR or RESPONDER sends an <iq>::<jingle> stanza with action type session-terminate. Its format is shown below.

```
<iq
   from="InitiatororResponderlocalpart@domainpart/UCAresourcepart">
   to="ResponderorInitiatorlocalpart@domainpart/MediaServerUCCDresourcepart">
   id="vendor generated value"
   type="set"
   <jingle
      xmlns="urn:xmpp:jingle:1"/>
      action="session-terminate"
      initiator="Initiatorlocalpart@domainpart/UCAresourcepart"
      sid="vendor generated value">
    <reason>
      <element from Table 3 of [XEP-0166] />
    </reason>
   </jingle>
</iq>
```

Resources for the STUN and TURN bindings will typically be released in short order since they are based on a short-term credential which is defined in [RFC-5389] and reproduced in part below.

> Short-Term Credential: A temporary username and associated password that represent a shared secret between client and server. Short-term credentials are obtained through some kind of protocol mechanism between the client and server, preceding the STUN exchange. A short-term credential has an explicit temporal scope.

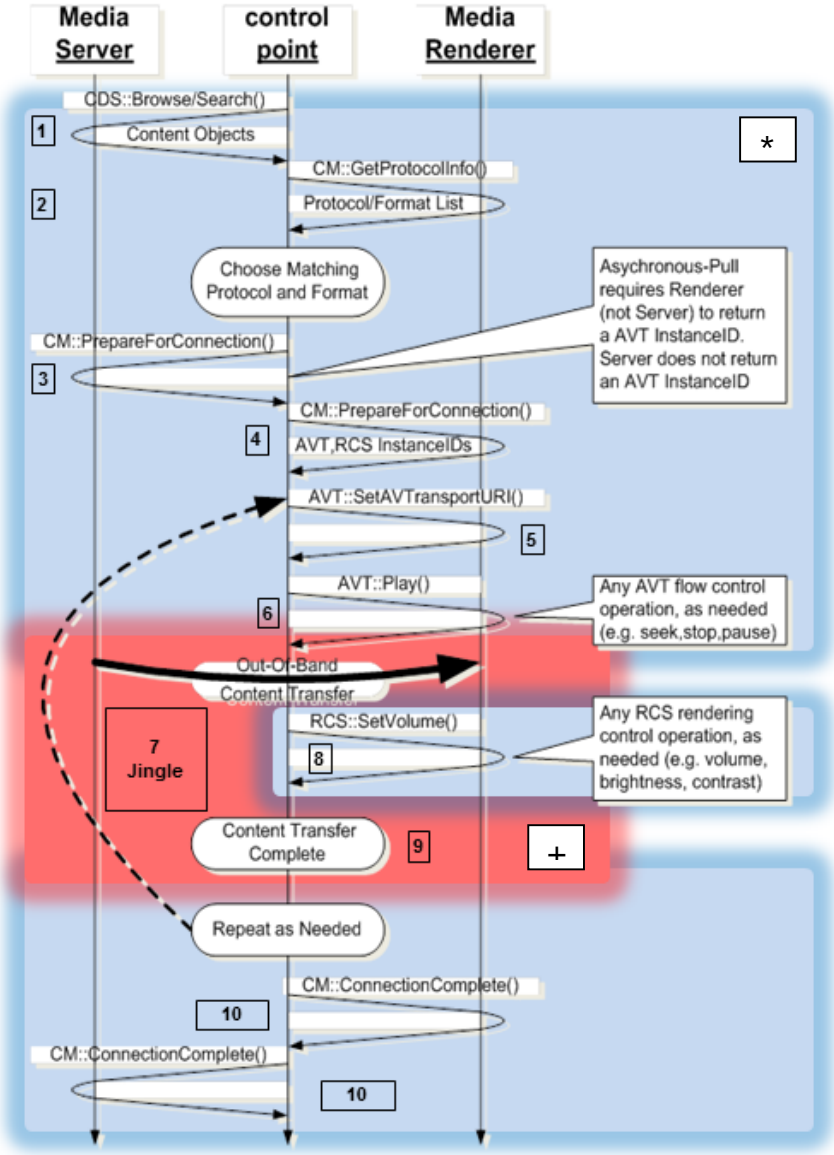
Therefore, STUN/TURN resources are not usable by other clients during a session and expire shortly after use is discontinued. Additionally, most NATs will be some form of (Port) Restricted Cone that limits incoming packet delivery to only IP addresses (and ports) that have been sent a packet by an endpoint behind the NAT. It is expected that Jingle endpoints that terminate the session properly according to ICE and take into account the security considerations in the related RFCs and XEPs should not have security or resource issues. Note that as WEBRTC becomes fully deployed much of the trusted infrastructure needed for Jingle will come along with it. In the meantime infrastructure components such as UCS, STUN, and TURN could be offered by such organizations as UPnP or the Open Interconnect Consortium or many vendors of UPnP cloud devices.

4 Theory of Operations

The 3-Box model interaction shown in Figure 4-1 provides a suitable example for illustrating the interactions of UPnP Cloud where Jingle is supported and streaming will occur over the public internet. For this example it is assumed that none of the endpoints - the UCCD MediaRender and MediaServer and the UCC-CP AV control point - are in the same local network. Also, in this example all three devices belong to the same user, `jeffrey@myloud.org`, however, the same basic call flow would work if the users were a mixed group for instance `jeffrey@mycloud.org` and `mary@theircloud.org`. In this example it is assumed UPnP Cloud Discovery, Description, and Event (PubSub) Subscription has already occurred (see [UDA] Annex C) and the call flow is picked up at the ContentDirectory service *Browse()* or *Search()* action invocation (see steps after figure). For this example the AV UCC-CP, MediaRenderer and MediaServer UCCDs have the Full JIDs:

- `jeffrey@upnp.org/urn:schemas-upnp-org:cloud-1-0:ControlPoint:1:ad93e8f5-634b-4123-80ca-225886a5c0e8`
- `jeffrey@upnp.org/urn:schemas-upnp-org:device:MediaRenderer:3:uuid:88509d0e-e8f5-80ca-4123-225886a50ee7`
- `jeffrey@upnp.org/urn:schemas-upnp-org:device:MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-636a323e7950`

Figure 4-1: 3-Box UPnP Cloud AV Streaming with Jingle Call Flow.



* - Blue background indicates UPnP actions sent via UPnP Cloud through the UCS.
+ - Red background indicates Out-of-Band Content Transfer in this case Jingle ICE-RTP.

In step [1]:

Jeffrey using his AV UCC-CP finds a piece of content on his UCCD MediaServer using UCA *ContentDirectory::Browse()* actions that he wants to play on his UCCD MediaRenderer (this content *item* is the same *<res>* as previously shown in 2.4).

In step [2]:

Jeffrey's AV UCC-CP using UCA invokes a *ConnectionManager::GetProtocolInfo()* action to see if the UCCD MediaRenderer advertises playback support for

`jingle-rtp-udp:*:video/mpeg:DLNA.ORG_PN=AVC_TS_NA_ISO`[12]

in its returned *SinkProtocolInfo* state variable which matches the content format he wants to play from the MediaServer; it does match.

In step [3] and [4]:

Jeffrey's AV UCC-CP using UCA issues a *ConnectionManager::PrepareforConnection()* action to the MediaServer/MediaRenderer UCCD to configure the UCCD MediaServer/MediaRender: The UCCD's *ConnectionManager::PrepareForConnection()* action (if implemented) informs the MediaServer and MediaRenderer that an outgoing/incoming connection is about to be made using the specified transfer protocol and data format that was previously selected. Depending on the selected transfer protocol, either the MediaServer or MediaRenderer will return an *AVTransport InstanceID*. This *InstanceID* is used in conjunction with the device's AVTransport Service (i.e. the device returning the *AVTransport InstanceID*) to control the flow of the content (e.g. *AVTransport::Play()*, *AVTransport::Stop()*, *AVTransport::Pause()*, *AVTransport::Seek()*, etc). Additionally, the MediaRenderer will return a Rendering Control *InstanceID* that is used by the control point to control the Rendering characteristics of the content. Note: Since *ConnectionManager*::*PrepareForConnection()* is an optional action, there may be situations in which either the MediaServer and/or MediaRenderer do not implement *ConnectionManager::PrepareForConnection()*. When this occurs and neither MediaServer nor MediaRenderer return an *AVTransport InstanceID*, the control point uses an *InstanceID*=0 to control the flow of the content. Refer to the ConnectionManager and AVTransport Service for details.

Below the UCC-CP indicates to the UCCD MediaServer that it will be connecting it to the UCCD MediaRender with a specific protocol, indicates the specific MediaRenderer and Peer ConnectionManager, and the direction of the connection relative to the MediaServer. The MediaServer responds that it can support more than one connection, in this case that it will be connection 2, and that it does not support the AVTransport or RenderingControl services (`-1` value).

`UCC-CP action to UCCD MediaServer`

```
PrepareForConnection(
   jingle-rtp-udp:*:video/mpeg:DLNA.ORG_PN=AVC_TS_NA_ISO,
   uuid:88509d0e-e8f5-80ca-4123-225886a50ee7/ConnectionManager,
   0utput
)
```

`UCCD MediaServer response to UCC-CP`

```
PrepareForConnection(
   2,
  -1,
  -1
)
```

Similarly, the UCC-CP can prepare the MediaRenderer for connection as shown below. In this case the MediaRenderer supports only one connection and the default "`0`" is used for all 3 instances (ConnectionManager, AVTransport, and RenderingControl).

`UCC-CP action to UCCD MediaServer`

```
PrepareForConnection(
   jingle-rtp-udp:*:video/mpeg:DLNA.ORG_PN=AVC_TS_NA_ISO,
   uuid:e70e9d0e-d9eb-4748-b163-636a323e7950/ConnectionManager,
   Input
)
```

`UCCD MediaServer response to UCC-CP`

[12] Note that the examples are using the suggested *protocolInfo* 1st field value of "`jingle-rtp-udp`" for this example.

```
PrepareForConnection(
   0,
   0,
   0
)
```

At this point the UCCD MediaServer and MediaRender can start allocating resources to connect and serve the particular content, for instance see if they are locally connected or start gathering IP address:port information for a Jingle session if not.

In step [5] and [6] and [7]:

The AV UCC-CP sets the URI of the specific piece of content that will be streamed and at this point the Jingle **session-initiate** <**iq**> is sent with the content item URI as described in 2.3. Once the initial component negotiation, security negotiation, connectivity steps have completed then the Jingle session can proceed to the ACTIVE state. If the MediaRenderer has sufficient buffer to receive content before an actual *AVTransport::Play()* action is invoked then the Jingle session can be negotiated to have the MediaServer start sending content at the full rate otherwise it can use RTP keep-alive to keep the setup NATs, STUN/TURN bindings configured. It the AVTransport service is on the UCCD MediaServer then the content send rate can be throttled directly; if it is on the MediaRenderer than the <bandwidth> element could be used by the INITATOR (MediaRender) Jingle client with a value of "0" prior to receiving a *Play()* from the UCC-CP and adjusted higher on the *Play()* invocation. The UCC-CP will be able to monitor progress of the streaming session via the subscribed **PubSub** events.

In step [8]

The UCC-CP can modify the streaming experience (muting, brightness, aspect, etc) by directly invoking RenderingControl service actions on the MediaRenderer UCCD using UCA actions. These actions should not affect the Jingle Session. The UCC-CP can also inoke AVTransport actions during the session. These will likely impact the Jingle session and need to be coordinated with specific Jingle related messages allowed during the ACTIVE state as shown in Figure 2-1.

In step [9]

The UCC-CP could decide to renderer more content using the same Jingle session setup instead, therefore instead of tearing down the session the UCCDs could keep the session alive and modify the content using Jingle session **content-add**, **content-remove**, **content-accept**, **content-reject** and **description-info** stanzas. For example, the MediaRenderer UCCD receives a AVTransport::Pause() action form the AV UCC-CP and needs to throttle the MediaServer media flow so it sends a Jingle description-info stanza to the MediaServer Jingle client with a <bandwidth> of value="0" to indicate that the MediaServer transition to RTP keep alives.

UCCD MediaRenderer Jingle to UCCD MediaServer

```
<iq
   from="jeffrey@upnp.org/urn:schemas-upnp-org:device:
         MediaRenderer:3:uuid:88509d0e-e8f5-80ca-4123-225886a50ee7"
   id="avt-pause-xu3bg810"
   to="jeffrey@upnp.org/urn:schemas-upnp-org:device:
       MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-636a323e7950"
   type="set">
   <jingle
      xmlns="urn:xmpp:jingle:1"
      action="description-info"
      initiator="jeffrey@upnp.org/urn:schemas-upnp-org:device:
                 MediaRenderer:3:uuid:88509d0e-e8f5-80ca-4123-225886a50ee7"
      sid="a73sjjvkla37jfea">
      <content
         creator="jeffrey@upnp.org/urn:schemas-upnp-org:device:
                  MediaRenderer:3:uuid:88509d0e-e8f5-80ca-4123-225886a50ee7"
```

```
        name="DLNAStream1">
        <description xmlns="urn:xmpp:jingle:apps:rtp:1" media="video">
           <payload-type id="33" name=MP2T" clockrate="90000">
           <parameter name="height" value="760"/>
        <parameter name="width" value="1080"/>
      </payload-type>
      <bandwidth type="AS">0</bandwidth>
      </description>
    </content>
  </jingle>
</iq>
```

The MediaServer accommodates and signals its accommodation by sending the result stanza.

```
<iq
   to= "jeffrey@upnp.org/urn:schemas-upnp-org:device:
          MediaRenderer:3:uuid:88509d0e-e8f5-80ca-4123-225886a50ee7"
   id="avt-pause-xu3bg810"
   from= "jeffrey@upnp.org/urn:schemas-upnp-org:device:
         MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-636a323e7950"
   type="result">
</iq>
```

If no other content is to be exchanges then the session can proceed to step [10].

In step [10]

If the optional *ConnectionComplete()* is supported on either or both of the UCCD endpoints then an explicit Jingle **session-terminate** can be invoked to close the connection and clean-up allocated resources, otherwise, the Jingle session-terminate will have to be derived from the media state, for example the end of the content has been reached; some additional, associated, implementation dependent time-outs could also be included.

5 Summary of Suggested Cloud Streaming Capabilities

Table 5-1 provides a short summary of capabilities (primarily XEPs) needed to implement the streaming scenarios presented in the document.

Table 5-1: Suggested UCC-CP and UCCD support for Cloud Based AV Streaming

Jingle Capabilities	UCC-CP	UCCD MediaServer	UCCD MediaRenderer
[XEP-0166], [XEP-0167] - Jingle	X[13]	X	X
[XEP-0176] - ICE-RTP-UDP	X	X	X
[XEP-0177] - Raw UDP	O[14]	O	O
[XEP-0234] - Jingle ICE-TCP File Transfer	X	Y[15]	Y
[XEP-0260] - Jingle SOCKS5	O	O	O
[XEP-0261] - Jingle IBB	X	X	X
[XEP-0293] - Jingle RTCP feedback	Z[16]	Z	Z
CloudTransport control point	X	Default	Default

[13] X = needed for typical Jingle support.

[14] O = implementers choice.

[15] Y = if serving or rendering HTTP content.

[16] Z = if supporting serving or rendering of RTP content.

6 References

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

[RFC-5245] - Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols. Available at http://tools.ietf.org/html/rfc5245

[RFC-5389] - Session Traversal Utilities for NAT (STUN). Available at http://tools.ietf.org/html/rfc5389

[RFC-5766] - Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN). Available at http://tools.ietf.org/html/rfc5766

[RFC-6062] - Traversal Using Relays around NAT (TURN) Extensions for TCP Allocations. Available at http://tools.ietf.org/html/rfc6062

[RFC-6120] - Extensible Messaging and Presence Protocol XMPP: Core. Available at http://tools.ietf.org/html/rfc6120

[RFC-6121] - Extensible Messaging and Presence Protocol XMPP: Instance Messaging and Presence. Available at http://tools.ietf.org/html/rfc6121

[RFC-6122] - Extensible Messaging and Presence Protocol XMPP: Address Format. Available at http://tools.ietf.org/html/rfc6122

[RFC-6544] - TCP Candidates with Interactive Connectivity Establishment (ICE). Available at http://tools.ietf.org/html/rfc6544

[XEP-0030] - Service Discovery, XMPP Standards Foundation, 1999-2015. Available at http://xmpp.org/extensions/xep-0030.html

[XEP0045] - Multi-User Chat. XMPP Standards Foundation, 1999-2015. Available at http://xmpp.org/extensions/xep-0045.html

[XEP-0060] - Publish-Subscribe, XMPP Standards Foundation, 1999-2015. Available at http://xmpp.org/extensions/xep-0060.html

[XEP-0166] - Jingle, XMPP Standards Foundation, 1999-2015. Available at http://xmpp.org/extensions/xep-0166.html

[XEP-0167] - Jingle RTP Sessions, XMPP Standards Foundation, 1999-2015. Available at http://xmpp.org/extensions/xep-0167.html

[XEP-0176] - Jingle ICE-UDP Transport Method, XMPP Standards Foundation, 1999-2015. Available at http://xmpp.org/extensions/xep-0176.html

[XEP-0177] - Jingle Raw UDP Transport Method, XMPP Standards Foundation, 1999-2015. Available at http://xmpp.org/extensions/xep-0177.html

[XEP-0234] - Jingle File Transfer, XMPP Standards Foundation, 1999-2015. Available at http://xmpp.org/extensions/xep-0234.html

[XEP-0260] - Jingle SOCKS5 Bytestreams Transport Method, XMPP Standards Foundation, 1999-2015. Available at http://xmpp.org/extensions/xep-0260.html

[XEP-0261] - Jingle In-Band Bytestreams Transport Method, XMPP Standards Foundation, 1999-2015. Available at http://xmpp.org/extensions/xep-0261.html

[XEP-0293] - Jingle RTP Feedback Negotiation, XMPP Standards Foundation, 1999-2015. Available at http://xmpp.org/extensions/xep-0293.html

[UDA] - UPnP Device Architecture, version 2.0, UPnP Forum, February 20, 2015. Available at: http://upnp.org/specs/arch/UPnPDA10_20000613.pdf. Latest version available at: http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v2.0.pdf.

[DP] - UPnP DeviceProtect:1 Service, UPnP Forum, February 4, 2011. Available at: http://www.upnp.org/specs/gw/UPnP-gw-DeviceProtection-v1-Service-20110224.pdf. Latest version available at: http://www.upnp.org/specs/av/UPnP-av-AVArchitecture-v2.pdf.

[AV-ARCH] - UPnP AV Architecture:2, UPnP Forum, March 31, 2013. Available at: http://www.upnp.org/specs/av/UPnP-av-AVArchitecture-v2-20130331.pdf. Latest version available at: http://www.upnp.org/specs/av/ContentDirectory-av-v4-Service.pdf.

[CDS4] - UPnP Content Directory:4 Service, UPnP Forum, June 30, 2015. Available at: http://www.upnp.org/specs/av/ContentDirectory-av-v4-Service-201150630.pdf. Latest version available at: http://www.upnp.org/specs/av/ContentDirectory-av-v4-Service.pdf.

[MS4] - UPnP MediaServer:4 Device, UPnP Forum, March 31, 2013. Available at: http://www.upnp.org/specs/av/MediaServer-av-v4-Device-20130331.pdf. Latest version available at: http://www.upnp.org/specs/av/MediaServer-av-v4-Device.pdf.

[MR3] - UPnP MediaRenderer:3 Device, UPnP Forum, March 31, 2013. Available at: http://www.upnp.org/specs/av/MediaRenderer-av-v3-Device-20130331.pdf. Latest version available at: http://www.upnp.org/specs/av/MediaRenderer-av-v3-Device.pdf.

[PROXY] - UPnP CloudProxy:1 Device, UPnP Forum July 1, 2013. Available at: http://www.upnp.org/specs/cloud/UPnP-cloud-CloudProxy-v1-Device-20151231.pdf. Latest version available at: http://www.upnp.org/specs/cloud/UPnP-cloud-CloudProxy-v1-Device.pdf.

[CPROXY] - UPnP CloudProxy:1 Service, UPnP Forum, December 31, 2015. Available at: http://www.upnp.org/specs/cloud/UPnP-cloud-CloudProxy-v1-Service-20151231.pdf. Latest version available at: http://www.upnp.org/specs/cloud/UPnP-cloud-CloudProxy-v1-Service.pdf.

[CTS] - UPnP CloudTransport:1 Service, UPnP Forum, December 31, 2015. Available at: http://www.upnp.org/specs/cloud/UPnP-cloud-CloudTransport-v1-Service-20151231.pdf. Latest version available at: http://www.upnp.org/specs/cloud/UPnP-cloud-CloudTransport-v1-Service.pdf.

[RAARCH] - UPnP Remote Access Architecture:2 Service, UPnP Forum, April 30, 2011. Available at: http://www.upnp.org/specs/ra/UPnP-ra-RAArchitecture-v2-20110430.pdf. Latest version available at: http://www.upnp.org/specs/ra/UPnP-ra-RAArchitecture-v2.pdf.