INTERNET-DRAFT
draft-cordell-success-00.txt

Cordell
BT

Nov 22, 1996
Expires: 22 May 1997

Simple Universal Call/Conference Establishment Sequence - (SUCCESS)

Status of this memo

Abstract

Currently in the Internet there are a number of call control protocols, each of them tailored to their own special applications. This includes SDP for session announcement, SIP and SCIP for session invitation and Q.931 used in H.323. None of these is likely to be turned into a generic call control protocol. Q.931 is limited to point-to-point calls. SDP and SIP do not include close down phases which are important if calls are being charged for on a timed basis or gateways are involved. Nor do they support supplementary services such as transfer. This proposal addresses these issues by defining a protocol based on a new conference control paradigm (referred to as the hello-hello paradigm) that can be used to create and control conferences from simple point-to-point calls, to large `broadcasts' and all call models in between. Both real-time peer-to-peer conversational models and client-server streaming models are catered for in the protocol so that all forms of real-time stream can feature in a conference.

Table of Contents

1. Introduction

This document describes a call setup procedure for use in an Internet environment. It allows flexible call setup, ranging from initiating a point-to-point call in a tightly coupled fashion, to a multicast conference announcement in a very loosely coupled fashion, and a number of conference models in between. It has also been designed with the intention of allowing gatewaying to other communications networks to be easily facilitated. The protocol operates over an unreliable datagram service such as the Internet's UDP service. Reliability of the protocol, when desired, is achieved by retransmission of messages adopting an algorithm similar to that of RTCP to select the retransmission interval.

The protocol is intended to amalgamate a number of the features of the IETF SDP, SIP and SCIP protocols and the ITU Q.931 protocol into a single unified Internet call setup protocol.

During the design of this protocol the main goals have been:

> To develop an open standard for seamless end-to-end multimedia communication independent of underlying network boundaries and transport technology. i.e. the protocol used in the Internet should translate cleanly and easily into call setup protocols used for ISDN and

ATM.

Users should have a consistent set of network services independent of the network providing the connection. i.e. facilities like transfer and hold should be supported (even if they are not implemented in a specific product).

The call setup protocol should enable the maximum potential and flexibility of the Internet infrastructure to be realised. Seamless migration between the various call models should also be supported directly at the protocol level. i.e. tightly coupled and loosely coupled conferences should not be explicitly differentiated at the protocol level, and migration between these modes should be possible as the conference evolves.

The inclusion of all media streams within a conference should be at the control of the call control protocol whether they be real-time conversational data, real-time live feeds, or pre-stored server fed streams. i.e. clients should invite and handle media streams within a conference in the same way they handle all other real-time feeds.

Support for capability negotiation should be supported, even in large multicast conferences where possible, so that systems will automatically select the best media transport system available, thus allowing effective exploitation of new and evolving coding technologies.

The protocol should be extensible for the purposes of adding new features to the standard protocol, and adding extensions for the purpose of evaluating new features. This should be done where possible without relying on a defined set of version numbers.

It has also been important to design a protocol that is rich and descriptive, thus enabling terminal designs that can enhance the user experience, but will at the same time collapse down to a very simple sub-set by ignoring certain fields that will allow basic terminals to be developed as initial product offerings.

Note that throughout this document the terms call and conference are used interchangeably. For the purposes of this document, it is NOT necessarily implied from the term call that the communication is point-to-point, and it is NOT necessarily implied from the term conference that the communication is multipoint.

2. Overview

To be suitable for use when setting up both point-to-point calls and multicast conferences the protocol effectively announces that its associated endpoint is in a call in the same way that a person uses `Hello' when greeting another person.  This approach differs to the Request and Acknowledge call setup style used in some protocols (which is equivalent to the `may I speak to you/yes you may' paradigm), and the billboard style advertisement used in other protocols (which is equivalent to the `big show on Friday' paradigm).

To provide tighter control where required (such as in the point-to-point case, or even for a small sub-set of a large multicast) the messages have one or more fields called Reply fields that allow them to specify who should acknowledge the message. In a multicast `broadcast' from a single point the Hello message might not contain a Reply field. In a multicast conference that contains a few core people, and then anybody else that wishes to listen in, the message would contain a number of Reply fields for each of the endpoints that had to be in the conference before the conference was worth proceeding with.

The protocol uses only five main message types. These are introduced here with a brief description. They are described further later on.

Hello   Used both to initiate a call and answer a call. Effectively it announces that an endpoint has entered into a point-to-point call or a multiparty conference.

Progress This message indicates the progress of a call being setup. This is intended primarily to give feedback to a user about the progress of call setup and does not result in any state changes. A number of progress message types exist, such as ringing, performing address lookup, transferring to POTS network etc. This message can be generated by intermediate points if they are involved in the call setup process. Feed back from the Progress message allows the user and application to know that the call/conference is progressing, thus preventing a user waiting for an answer from a terminal that is not switched on.

Bye   This message gives an endpoint the option to signal that an it has left, or is leaving a call/conference.

Byebye Is sent to acknowledge a Bye from an endpoint. The use of this is described further below.

Feature Used for additional signalling such as transferring calls and putting people on hold. The minimum support required for this message is to identify when a message has been sent to you, and respond with the notSupported element of the message. Much of this processing is identical to the other messages, and so this does not represent a significant burden. The use of this message is intended to be the place where extra functionality is added into the protocol. The idea is to have optional bolt-on services/protocols into this message. One such bolt-on that has already been specified is for control of real-time streams supplied by servers.

The main elements of these messages as far as the protocol is concerned are the 'from', 'to', 'reply', and 'replyAck' fields. The purpose of the first two should be quite obvious. The third field indicates which terminals should send a reply in response to the message sent. A reply is indicated by putting the name of the terminal that is being replied to in the replyAck field.

The main rules of the protocol are that if you receive a Hello message with your endpoint mentioned in the Reply field, you should send either a Progress message or a Hello message with the name of the terminal you are responding to indicated in the replyAck field and the From field set. When a hello message is received with your endpoint mentioned in the replyAck field, you should send a Hello message that does not include the sender in either the reply or replyAck fields. If you receive a Bye message with your endpoint mentioned in the Reply field, you should send a Byebye message with the From and To fields set. Further description is included below to show how these messages and rules can be used to setup and cleardown conferences.

3. Detailed Description

The above outlines the principles of operation. This section adds more detail.

3.1. Messages

This section indicates the sorts of fields that the various messages contain. See the section on message encoding to see how the messages are encoded on the line.

Note that the most important fields are `cID`, `from`, `to', `reply` and `replyAck`.

```
Hello     ::= SEQUENCE
     {
     cID             GUID,
     from            UserAddress,
     to              SET OF UserAddress OPTIONAL,
     reply           SET OF UserAddress OPTIONAL,
     replyAck        SET OF UserAddress OPTIONAL,
     respondTo       NetAddress OPTIONAL,
     refreshX3       INTEGER ( 1 .. 65535 ) OPTIONAL,
     description     Text OPTIONAL,
     display         Text OPTIONAL,
     time            SET OF Time OPTIONAL,
     userInfo        SET OF UserAddress OPTIONAL,
     capno           INTEGER (0..65535 ) OPTIONAL,
     caps            SET OF Capability OPTIONAL,
     sendno         INTEGER (0..65535 ) OPTIONAL,
     sending        SET OF Property OPTIONAL,
     ...
     }
```

cID A unique number specifying the conference.

from A unique name specifying who the message is from. This specifies the participant at the protocol level. This information should remain consistent throughout the conference.

to The primary intended recipient of the message. Allows messages to be addressed to a sub-set of the conference and for handling by proxy or location service.

reply Who should reply to this message. If an endpoint finds an alias for itself in this list, it must respond with the specified alias as opposed to another of its aliases.

replyAck Indicates the terminals to which a reply is being sent in response to an earlier reply message.

respondTo The address (possible multicast address) to which all responses should be sent. This allows a terminal to send an invitation to a remote terminal using point-to-point addressing, but have the remote terminal respond to the messages to the conference multicast address.

refreshX3 Indicates the time in seconds by which a minimum of three subsequent hello messages should have been received. If one or more hello messages (only one is required) have not been received in this period, the receiver can assume the session has ended. This allows a receiver to know that a session has ended without explicit notification. An interval of three refreshes is specified to allow for lost packets.

description Short description of the session.

display   Information that might be presented to a remote user about the state of this endpoint.

time   When the session should take place. If this field is not present, then the conference is considered to be now.

userInfo Additional information about the person sending the message.

capno   Specifies the instance of caps set specified in the caps part of the message. If the sender modifies the data contained in the caps section, then it should increment the value contained in this field by 1. This is to remove the need for the receiver to continually parse the caps section looking for changes.

caps   The set of capabilities that this terminal can receive.

sendno   Specifies the instance of sending parameter specified in the sending part of the message. If the sender modifies the data contained in the sending section, then it should increment the value contained in this field by 1. This is to remove the need for the receiver to continually parse the sending section looking for changes.

sending   Describes information about a stream. Its main use is to describe the actual streams which are being sent by a sender.

```
Progress  ::= SEQUENCE
     {
     cID            GUID,
     from           UserAddress,
     to             SET OF UserAddress OPTIONAL,
     phase          ProgressPhase,
     fromEndpoint   BOOLEAN,
     capno          INTEGER (0..65535 ) OPTIONAL,
     caps           SET OF Capability OPTIONAL,
     display        Text OPTIONAL,
     ...
     }
```

cID A unique number specifying the conference.

from A unique name specifying who the message is from.

to The primary intended recipient of the message.

phase Progress status code indicating things like looking up address, ringing etc.

fromEndpoint Set to TRUE if message generated by t

capno Specifies the instance of caps set specified in the caps part of the message. If the sender modifies the data contained in the caps section, then it should increment the value contained in this field by 1. This is to remove the need for the receiver to continually parse the caps section looking for changes.

caps The set of capabilities that this terminal can receive. By putting caps in the Progress message it is possible to do decisions on the conference caps used prior to the conference starting.

display Information that might be presented to a remote user about the state of this endpoint.

```
Bye  ::= SEQUENCE
     {
     cID            GUID,
     from           UserAddress,
     to             SET OF UserAddress OPTIONAL;
     reply          SET OF UserAddress OPTIONAL,
     reason         ByeReason OPTIONAL,
     display        Text OPTIONAL,
     ...
     }
```

cID A unique number specifying the conference.

from A unique name specifying who the message is from.

reply Who should reply to this message.

reason Why the connection was closed. These might consist of: Normal, Busy, Unknown address, Ambiguous address, Redirect, Alternative service (see SIP), Join conference, No resources, Unspecified, etc.

display Information that might be presented to a remote user about the state of this endpoint.

```
Byebye         ::= SEQUENCE
     {
     cID       GUID,
     from      UserAddress,
     to        SET OF UserAddress,
     display   Text OPTIONAL,
     ...
     }
```

cID A unique number specifying the conference.

from A unique name specifying who the message is from.

to The primary intended recipient of the message. Indicates who the Byebye message is in response to.

display Information that might be presented to a remote user about the state of this endpoint.

```
Feature     ::= SEQUENCE
     {
     cID    GUID,
     from   UserAddress,
     to     UserAddress OPTIONAL,
     fID    FeatureSeqNo,
     mode   CHOICE
            {
            reqAck            ServiceType,
            reqNoack          ServiceType,
            ack               NULL,
            querySupported    ServiceList,
            isSupported       NULL,
            notSupported      NULL,
            ...
            },
     ...
     }
```

cID   A unique number specifying the conference.

from   A unique name specifying who the message is from.

to   The primary intended recipient of the message.

reqAck   Request a service that should be acknowledged

reqNoack Request a service that should not be acknowledged

ack   Acknowledges a feature. The FeatureSeqNo shall correspond to the FeatureSeqNo set in the request message.

querySupported   Asks the remote end if a feature is s message.

isSupported   Sent in response to a querySupported the request message.

notSupported   Signals that a requested feature is n correspond to the FeatureSeqNo set in the request message.

3.2. Message Sub Types

```
-- The root message element

SUCCESSV1 ::= CHOICE
     {
     hello          Hello,
     progress       Progress,
     bye       Bye,
     byebye         ByeBye,
     feature        Feature,
     ...
     }

GUID      ::= OCTET STRING ( SIZE(16) )
Text      ::= BMPString( SIZE(0..511) )

UserAddress    ::=  CHOICE
     {
     email          Text,
     locator        Text,     -- Name meaningful to
                              -- location service
     system         Text,
     url            Text,     -- For identifying files on
                              -- servers
     ipdotted       Text,     -- IP dotted notation
     e164           SEQUENCE OF SEQUENCE
                        -- Allow multiple E.164 numbers per
                        -- single destination
                    {
                    extension Text,
                    remoteAddr     Text OPTIONAL,
                    remoteSubAddr  Text OPTIONAL
                    ...
                    },
     fax            Text,
     title          Text,  --e.g. `Director of BT Labs'
     tag            GUID,  --machine assigned address
     commonName     Text,
     role           Role,
     network        NetAddress,
     ...
     }
```

```
Role      ::= CHOICE
     {
     chairperson    NULL,
     secretary      NULL,
     speaker        INTEGER(0..65535),
     panel          INTEGER(0..65535),
     controller     NULL,
     ...
     }

NetAddress     ::= CHOICE
     {
     ip4  SEQUENCE
          {
          ip        OCTET STRING ( SIZE(4) ),
          port      INTEGER( 0..65535 ) OPTIONAL,
          ttl       INTEGER( 0..255 ) OPTIONAL,
          service   CHOICE { UDP NULL, TCP NULL, ...} OPTIONAL,
          route     SEQUENCE OF OCTET STRING SIZE(4) OPTIONAL
                                    -- for source routing
          },
     ip6  SEQUENCE
          {
          ip        OCTET STRING ( SIZE(16) ),
          port      INTEGER( 0..65535 ) OPTIONAL,
          ttl       INTEGER( 0..255 ) OPTIONAL,
          service   CHOICE { UDP NULL, TCP NULL, ...} OPTIONAL,
          route     SEQUENCE OF OCTET STRING SIZE(16) OPTIONAL
                                  -- for source routing
          },
     ...
     }

ProgressPhase  ::= CHOICE
     {
     locating       NULL,     --proxy of some description
                              --is locating user
     placed         NULL,     --Users terminal has
                              --received call indication
     ringing        NULL,     --User terminal is ringing
     gatewaying     NULL,     --Transferring to POTS or
                              --ISDN
     willattend     NULL,     --Signals that the user
                              --will attend a conference taking
                              --place in the future that they
                              --have been invited to attend
     ...
     }
```

```
ByeReason ::= CHOICE
     {
     normal         NULL,
     unauthorized   NULL,
     deferred       NULL,     -- Do not disturb
     callback       NULL,     -- User will callback
                              -- when free
     busy           NULL,
     feature        NULL,     -- Bye due to
                              -- signalled feature
     unknown        NULL,     -- Person or file not known
     ambiguous      NULL,     -- Address is incomplete
     deflection     SEQUENCE
                    {
                    cID        GUID OPTIONAL,
                    user       UserAddress,
                    conference BOOLEAN OPTIONAL,
                    display    Text OPTIONAL,
                    ...
                    },   --must do re-routing end-less
                         --loop detection
     noCaps         NULL,-- No common capabilities
     noLocation     NULL,
     noNetResources NULL,
     noSysResources NULL,
     ...
     }

Time      ::= SEQUENCE
     {
     first     INTEGER(0..4294967295),  --NTP seconds
                                --time of first showing
     duration  INTEGER(0..4294967295),
     repeat    SEQUENCE OF
               {
               delay    INTEGER(0..429496795), --seconds
               times    INTEGER(0..255),       --how many
                                               --repeats
               ...
               } OPTIONAL,
     ...
     }
```

```
Capability     ::= CHOICE
     {
     --Video modes
     h261      H261,
     h262      H262,
     h263      H263,

     --Audio modes
     gsm       AudioParameters,
     g711Alaw  AudioParameters,
     g711Ulaw  AudioParameters,
     g722-64k  AudioParameters,
     g722-56k  AudioParameters,
     g722-48k  AudioParameters,
     g723      SEQUENCE
               {
               maxAI-sduAudioFrames   INTEGER( 1..256 ),
               silenceSuppression     BOOLEAN,
               address                NetAddress,
               setNum                 SET OF INTEGER(0..255),
               payloadtype            INTEGER( 0..127) OPTIONAL,
               description            Text OPTIONAL
               },
     g728      AudioParameters,
     g-dsvd    AudioParameters,

     --Data modes
     t120      ControlParameters,
     sccp      ControlParameters,

     --Control modes
     h323      H323Parameters,
     ...
     }

AudioParameters          ::= SEQUENCE
     {
     maxFPP         INTEGER(1..2048),
                            --Max frames per packet
     address        NetAddress,
     setNum         SET OF INTEGER(0..255),
     payloadtype    INTEGER( 0..127) OPTIONAL,
     ssrc           INTEGER( 0..2^32-1 ) OPTIONAL,
                            -- Only used in sending field
     description    Text OPTIONAL,
     ...
     }
```

```
ControlParameters   ::= SEQUENCE
     {
     address        NetAddress,
     setNum         SET OF INTEGER(0..255) OPTIONAL,
     ...
     }

H323Parameters      ::= SEQUENCE
     {
     crv       INTEGER( 1..65535 ),
     type      EndpointType, -- See H.225 for definition
     activeMC  BOOLEAN,
     conferenceGoal CHOICE
               {
               create    NULL,
               join      NULL,
               invite    NULL,
               ...
               }  OPTIONAL,
     h245      NetAddress OPTIONAL,
     ...
     }

-- The encoding for the H.261, H.262 and H.263 modes are based on H.245
H261 ::= SEQUENCE
     {
     qcifMPI        INTEGER( 1..4 ) OPTIONAL,
     cifMPI         INTEGER( 1..4 ) OPTIONAL,
     maxBitRate     INTEGER( 1..19200 ),
     stillImage     BOOLEAN,       --H.261 Annex D

     address        NetAddress,
     setNum         SET OF INTEGER(0..255) OPTIONAL,
     payloadtype    INTEGER( 0..127) OPTIONAL,
     ssrc           INTEGER( 0..2^32-1 ) OPTIONAL,
                           -- Only used in sending field
     description    Text OPTIONAL,
     ...
     }
```

```
H262 ::= SEQUENCE        --MPEG1
     {
     profileAndLevel-SPatML         BOOLEAN,
     profileAndLevel-MPatLL         BOOLEAN,
     profileAndLevel-MPatML         BOOLEAN,
     profileAndLevel-MPatH-14       BOOLEAN,
     profileAndLevel-MPatHL         BOOLEAN,
     profileAndLevel-SNRatLL        BOOLEAN,
     profileAndLevel-SNRatML        BOOLEAN,
     profileAndLevel-SpatialatH-14 BOOLEAN,
     profileAndLevel-HPatML         BOOLEAN,
     profileAndLevel-HPatH-14       BOOLEAN,
     profileAndLevel-HPatHL         BOOLEAN,
     videoBitRate                   INTEGER (0.. 1073741823)
                                         OPTIONAL, -- units 400 bits/sec
     vbvBufferSize                  INTEGER (0.. 262143)
                                         OPTIONAL, -- units 16384 bits
     samplesPerLine                 INTEGER (0..16383)
                                         OPTIONAL, -- units samples/line
     linesPerFrame                  INTEGER (0..16383)
                                         OPTIONAL, -- units lines/frame
     framesPerSecond                INTEGER (0..15)
                                         OPTIONAL, -- frame_rate_code
     luminanceSampleRate            INTEGER (0..4294967295)
                                         OPTIONAL, -- units samples/sec

     address                        NetAddress,
     setNum                         SET OF INTEGER(0..255) OPTIONAL,
     payloadtype                    INTEGER( 0..127) OPTIONAL,
     ssrc                           INTEGER( 0..2^32-1 ) OPTIONAL,
     description                    Text OPTIONAL,
     ...
     }
```

```
H263 ::= SEQUENCE
     {
     sqcifMPI            INTEGER (1..32) OPTIONAL,
                                -- units 1/29.97 Hz
     qcifMPI             INTEGER (1..32) OPTIONAL,
                                -- units 1/29.97 Hz
     cifMPI              INTEGER (1..32) OPTIONAL,
                                -- units 1/29.97 Hz
     cif4MPI             INTEGER (1..32) OPTIONAL,
                                -- units 1/29.97 Hz
     cif16MPI            INTEGER (1..32) OPTIONAL,
                                -- units 1/29.97 Hz
     maxBitRate          INTEGER (1..19200),
                                -- units 100 bits/s
     unrestrictedVector  BOOLEAN,
     arithmeticCoding    BOOLEAN,
     advancedPrediction  BOOLEAN,
     pbFrames            BOOLEAN,
     hrd-B               INTEGER (0..524287) OPTIONAL,
                                -- units 128 bits
     bPPmaxKb            INTEGER (0..65535) OPTIONAL,
                                -- units 1024 bits

     address             NetAddress,
     setNum              SET OF INTEGER(0..255) OPTIONAL,
     payloadtype         INTEGER( 0..127) OPTIONAL,
     ssrc                INTEGER( 0..2^32-1 ) OPTIONAL,
     description         Text OPTIONAL,
     ...
     }

FeatureSeqNo   ::= INTEGER( 0..255 )

ServiceList    ::= CHOICE
     {
     call           NULL,
     authen         NULL,
     message        NULL,
     assignRole     NULL,
     rtsp           NULL,
     apps           NULL,
          ...
     }
```

```
ServiceType    ::= CHOICE
     {
     call           CallControl,
     authen         Authentication,
     message        SEQUENCE OF Text,
     assignRole     Role,
     rtsp           NetAddress,
     apps           Appshare,
     ...
     }

CallControl    ::= CHOICE
     {
     hold      NULL,
     holdack   NULL,
     holdrej   NULL,
     resume    NULL,
     resumeack NULL,
     resumerej NULL,
     transfer  SEQUENCE
               {
               cID        GUID OPTIONAL,
               user       UserAddress,
               conference BOOLEAN OPTIONAL,
               display    Text OPTIONAL,
               ...
               },
     transferack    NULL,
     transferrej    NULL,
     ...
     }

Authentication ::= CHOICE
     {
     challenge OCTET STRING SIZE( 0..64 ),
     cresponse OCTET STRING SIZE ( 0..64 ),
     ...
     }

Appshare  ::= CHOICE
     {
     reqList        NULL,
     list           SET OF Application,
     reqAddr        Application,
     addrAck        NetAddress,
     addrRej        NULL,
     ...
     }
```

```
Application    ::= CHOICE
     {
     t126                NULL,  -- Example
     word6.microsoft.com NULL,  -- Example
     notes.lotus.com     NULL,  -- Example
     ...
     }


Property            ::= SEQUENCE
     {
     stream         SET OF Capability,
     title          Text OPTIONAL,
     director       SET OF Text OPTIONAL,
     producer       SET OF Text OPTIONAL,
     actor          SET OF Text OPTIONAL,
     actress        SET OF Text OPTIONAL,
     created        Time OPTIONAL,
     duration       INTEGER( 0..2^32 ) OPTIONAL,
                            -- in milliseconds
     fastfrwd       INTEGER( 1..255 ) OPTIONAL,
                            -- Max fast frwd factor
     rewind         INTEGER( 1..255 ) OPTIONAL,
                            -- Max rewind factor,
     pause          BOOLEAN OPTIONAL,
     nudgeFrwd      BOOLEAN OPTIONAL,   -- single frame advance
     nudgeBack      BOOLEAN OPTIONAL,   -- single frame back
     live           BOOLEAN OPTIONAL,
     indexable      BOOLEAN OPTIONAL,
     ...
     }
```

3.3. Event processing

This section gives an example of the sequences that take place for each of the main events. As mentioned above, it is intended mainly for illustrative purposes.

For clarity, each event is presented in the form of C style pseudo-code. Due to the detailed nature of this description, its accuracy can not be guaranteed, and it might change in future versions.

The principle of the pseudo-code is that in a conference there a set of terminals that you want in the conference and a set of terminals that want you in the conference. As the conference evolves, this information is stored in two lists, 'my-reply' and 'reply-to' respectively. As much of this information is multicast, information can also be obtained on other terminals in the conference that you are not directly interested in. When Hello messages are sent, you copy the contents of the my-reply list to the message reply field, and the reply-to list to the replyAck field. The message is then sent to the super-set of the my-reply list and the reply-to list. For these two lists, as you receive replys from the specified endpoints, you remove them from the list appropriately (see Hello pseudo code below). If you receive a Hello message with your name in the reply then you add the name of the sender to the reply-to list. When the conference is stable, both lists should be empty. The frequency with which Hello messages are sent is controlled by two timers, Tfast and Tslow. Tfast is used as the time base to generate hello messages when the conference is undergoing a state change from the terminal perspective, i.e. when either the 'my-reply' contains entries marked as not progressing or the 'reply-to' lists is not empty. Tslow is used as the generator of hello messages when the conference is stable from the point of view of the endpoint, or the use of the Tfast timer doesn't seem to be progressing the conference state.

The result of this is that a three way handshake of Hello messages is set up, that can be interrupted a Progress message. Three stages are required (as opposed to two) because on receiving a progress message (from the remote user rather than an intermediary such as a proxy) the invitor will switch to using a slower timer for generating Hello messages. This does not allow for suitable response when the remote user answers the call as the Hello message generated in this instance may get lost. If this were to happen, the invitor would not send another Hello message inviting a response for many seconds, hence the invitor would not know that the remote user had entered the call. Therefore, rather than waiting for another Hello message, the remote user takes responsibility for ensuring that the invitor is aware they are in the conference by repeatedly sending Hello messages with the invitor indicated in the replyAck field until the invitor responds by sending a Hello message with the remote user absent from its reply list. Note that it is important to switch to using the slower timer when a Progress message is received as it may take a remote user many minutes to answer a call, during which time it is unacceptable to send multiple Hello messages at a high repetition rate.

A third list ('interested-in') stores the total of the 'my-reply' list and 'reply-to' list. This is used to copy to the 'my-reply' list when the conference is being closed down, thus informing all those that invited you and all those that you invited, that you are leaving the conference. (N.B. in practice these lists would probably be implemented as one list with a set of flags, but it's easier to describe in this way). A final detail on top of all this is whether the user is in or out of the conference, whether they are listen-only, or whether they have expressed an explicit desire to not be in the conference (e.g. they were in, but have since left). This generally affects how the reception of the hello and bye messages are handled.

In addition to the two timer mentioned above, there are two other timers, Trefresh and Tleaving. Tleaving generates Bye messages when the conference is being closed down. Its characteristics will probably be much the same as Tfast (but will expire after N time-outs rather than switching to Tslow). Trefresh is aimed at picking up terminals that have silently left the conference or failed. A field in the hello message indicates the period over which the sender intends to send 3 more hello messages. Trefresh is started, and all endpoints in the 'active-endpoint' list are marked as 'not refreshed'. As each hello message comes in, the endpoint that it is from is marked as refreshed. Also, a variable collects the maximum value presented in any of the hello messages refresh field during the refresh period. When Trefresh expires, it goes through the 'interested-in' list and knocks out all the endpoints that haven't refreshed thus assuming they have left the conference. Trefresh is then restarted with the value that has been calculated as the maximum refresh time.

Receive Hello:

```
   if( refresh time > auto refresh time )
      Update auto refresh time;
   Update `active-endpoints' list and set endpoint
                 refreshed flag;

   if( message directed to me or to all )
      {
      if( User-mode is active or listening )
         {
         // IF sender asking me to reply
         if( I'm in `message-reply' list && sender not in
                        `reply-to' list )
            {
            Add sender to `reply-to' list;  // Hello will be
                                            // sent later
            Set User-mode to active;
            }

         // IF sender acknowledges my reply
         if( sender in `reply-to' list && I'm not in
                        `message-reply' list )
            Remove sender from `reply-to' list;

         // IF sender responds to my reply request
         if( sender in `my-reply' list )
            Remove sender from `my-reply' list;

         if( User-mode is active )
            {
            if( `my-reply' list  does not contain any
                        entries marked as not progressing &&
                        `reply-to' list is empty &&
                        I'm not in 'message-replyAck' list )
               Ensure Tslow is running and Tfast is not;
            else
               Ensure Tfast is running and Tslow is not;
            }
         }

      else if( user not yet in conference )
         {
         Inform user of conference;
         if( I'm in `message-reply' list )
            {
            Add sender to `reply-to' list;
            Add sender to `interested-in' list;
```

```
                 Progress message;
             }
          }
```

```
      else if( user indicated not interested in
                    conference )
         if( I'm in `message-reply' list )
            Send Bye message with appropriate reason;
      }
```

Receive Progress:

```
   Record state against terminal;
   Inform user conference state changed;
   if( message is from sender [as opposed to an
                    intermediary] )
      Mark sender as progressing in 'my-reply' list;
```

Receive Bye:

```
   Remove endpoint from active-endpoints list;

   if( sender in `interested-in' list )
      {
      Remove from `interested-in' list;
      if( sender in `my-reply' list )
         {
         Remove sender from `my-reply' list;
         if( User-mode is active )
            {
            if( reason specifies alternative address )
               {
               Put new address in `interested-in' list;
               Put new address in `my-reply' list;
               if( Tslow is running )
                  Cancel Tslow timer;
               if( Tfast is not running )
                  Start Tfast and reset retransmission count;
               }
            }

         // ELSE allow for both endpoints to say bye at same time
         else if( User-mode is leaving )
            {
            if( `my-reply' list empty )
               {
               Inform user;
               Stop Tleaving;
               }
            }
         }
      }

   if( I'm in `message-reply' list )
      Send ByeBye message;
```

Receive ByeBye:

```
   if( message directed to me )
      {
      Remove remote sender from `my-reply' list;
      if( `my-reply' list is empty )
         {
         Inform user;
         Stop Tleaving;
         }
      }
```

User initiates call:

```
   Select conference ID;
   Put desired endpoints in `my-reply' list and mark as
                    not progressing;
   Put desired endpoints in `interested-in' list;
   Send Hello message to all endpoints in 'interested-in'
                    list copying 'my-reply' list to message
                    'reply' field, and copying 'reply-to'
                    list to message 'replyAck' field;
   Start Tfast and reset retransmission count;
   Set User-mode to active;
```

User invites new endpoints:

```
   Put desired endpoints in `my-reply' list and mark as
                    not progressing;
   Put desired endpoints in `interested-in' list;
   Send Hello message to all endpoints in 'interested-in' list
                    copying 'my-reply' list to message 'reply'
                    field, and copying 'reply-to' list to message
                    'replyAck' field;
   if( Tslow is running )
      Cancel Tslow timer;
   if( Tfast is not running )
      Start Tfast and reset retransmission count;
   Set User-mode to active;
```

User answers call:

```
   if( `reply-to' list is not empty )
      {
      Send Hello message  to all endpoints in
                    'interested-in' list copying 'my-reply'
                    list to message 'reply' field, and copying
                    'reply-to' list to message 'replyAck' field;
      if( Tslow is running )
         Cancel Tslow timer;
      if( Tfast is not running )
         Start Tfast and reset retransmission count;
      Start auto refresh timer (Trefresh) and set next period
                    time to zero;
      Set User-mode to active;
      }
   else
      Set User-mode to listening;
```

User leaves call:

```
   Stop Tfast and Tslow;
   Copy `interested-in' list to `my-reply' list;
   if( `my-reply' list not empty )
      {
      Send Bye message with Reply fields set and
                    appropriate disconnect reason;
      Initiate Bye closing retransmission timer logic (Tleaving);
      Reset Bye closing retransmission count (Nleaving);
      }
   else
      {
      if( it is desired to signal this endpoint leaving conference )
         Send Bye with Reply list empty and appropriate
                 disconnect reason;
      }
```

Timer Tfast times out:

```
   Decrement fast retransmission count (Nfast);
   if( retransmission count is not zero )
      {
      Send Hello message  to all endpoints in 'interested-in'
                    list copying 'my-reply' list to message
                    'reply' field, and copying 'reply-to'
                    list to message 'replyAck' field;
      if( `my-reply' list  does not contain any entries
                    marked as not progressing && `reply-to'
                    list is empty &&
                    I'm not in 'message-replyAck' list )
         Ensure Tslow is running and Tfast is not;
      else
         Ensure Tfast is running and Tslow is not;
      }
   else
      {  // Call is failing to progress
      if( `active-endpoints' list is not empty )
         Start TSlow;
      else
         {
         Inform user;
         Send Bye message with empty Reply list;
         }
      }
```

Timer Tslow times out:

```
   Send Hello message to all endpoints in 'interested-in'
                    list copying 'my-reply' list to message
                    'reply' field, and copying 'reply-to' list
                    to message 'replyAck' field;
   Re-evaluate and restart Tslow;
```

Timer Tleaving times out:

```
   if( `my-reply' list is not empty )
      {
      Send Bye message with Reply list;
      Decrement re-transmission count:
      if( retransmission count is not zero )
         Re-evaluate and restart Tleaving;
      else
         inform user;
      }
   else
      Inform user;
```

Timer Trefresh times out:

```
   Remove items from `interested-in' list, `my-reply'
                    list and `reply-to' list that have
                    not been marked as refreshed;
   if( next period refresh time is not zero )
      {
      Re-start Trefresh;
      Clear next period refresh count to zero;
      }
```

3.4. Main Information

3.4.1. Per conference information:

User-mode Whether the user is not in conference, only listening to the conference, actively involved in the conference, or not interested in the conference.

interested-in list Stores the list of endpoints that that this endpoint has either replied to or asked for replys from during the conference.

my-reply list Implemented as part of interested-in list. Stores the list of endpoints this endpoint wants to receive a reply from.

reply-to list Implemented as part of interested-in list. Stores the list of endpoints this endpoint should reply to.

endpoint-list List of all the terminals in the conference, or as many as the application is prepared to store.

Mcast-Address The call control session multicast address. This may not be used in some circumstances.

3.4.2. Information stored per endpoint in the interested-in list:

Quantity	Type	Description
name	String and type	The name of the endpoint.
my-reply	Flag	Indicates whether this endpoint wants a reply from the remote endpoint
reply-to	Flag	Indicates whether this endpoint should send a reply to the remote endpoint
progressing	Flag	Set to FALSE when an endpoint is initially invited to a call. When a Progress or Hello message is received from the endpoint, the flag is set to TRUE.
refreshed	Flag	Indicates whether the remote endpoint has sent a new Hello message within the refresh period
address	IP Addr/Port	The address and port to send messages to the remote endpoint. Maybe a unicast or multicast address depending on the conference phase and type
go-mcast	Flag	If True, indicates that if a terminal was invited to a conference using unicast, it should be signalled to use the conference multicast address. When a reply from the endpoint has been received on the conference multicast address, the address field above will be changed to the conference multicast address.
last-cap-no	Integer	The number of the last capability set sent by the terminal.
last-send-no	Integer	The number of the last send no set by the sender.

3.5. Timers

The protocol defines a numbers of timers. These are described here.

Timer value	Repeats	Description
Tfast	Nfast	Used as the time base to generate Hello messages when the endpoint is changing its membership lists
Tslow	Infinite	Used as the time base to generate Hello messages when the endpoint has a stable membership list
Tleaving	Nleaving	Used as a timebase to generate Bye messages when an endpoint is leaving a conference
Trefresh	Infinite	Used to detect endpoints that have silently left a conference

4. Capabilities

The capabilities in the hello message allows the sender of the message to specify media that the receiving endpoints can transmit. In addition to standard audio, video, and data capabilities, control capabilities are defined. This allows a protocol to be used on top of this protocol for setting up media streams such as H.323, SCCP and T.120.

Capabilities do not need to be present in every Hello message sent. If they are not present the previously specified capabilities are taken to be still valid. If one or more capabilities are changed, then a complete set of capabilities needs to be specified, thus overwriting the previous set. When the capability set is changed the sender should increment the capno parameter sent in the hello message to let the receiver know that a change has been made without the receiver having to parse the whole message.

When defining the set of capabilities that can be received, each declared capability is assigned to one or more 'sets'. These sets are numbered zero to 255. A maximum of one mode can be active from a given set at any one time. Thus, if GSM and G.723 are assigned to the same set, only one of them can be active at a time. If **G.723 is defined in two sets, then two G.723 streams can** be used simultaneously perhaps with different languages. This is thought to be the simplest mechanism possible that allows multiple options to be specified for multiple streams of the same media type. To specify more complicated capability sets, higher layer protocols such as SCCP, H.245, or T.120 should be used.

The method of defining proprietry extensions defined under the Use of the Feature Message can also be used to define proprietry codecs in the capability sets.

Note that when inviting a new terminal into a conference (i.e. when the hello message specifies a reply), the capabilities expressed should be that of the inviting terminals view of the conferences aggregated common capabilities and not solely those of the inviting terminal.

It is recommended that a mandatory set of base capabilities be defined that must be supported by all terminals. This will ensure that there will always be some degree of compatibility. An example is G.723, and if video is present, QCIF H.261. This base set would allow effective use over dial-up modem links.

This scheme works well for point-to-point cases and for large multicasts where negotiation is not allowed or not possible. However, it is desirable to extend the scheme such that effective mode negotiation can take place for at least a dozen terminals. This requires further work. Currently negotiation is looked upon as a two stage process; finding the common capabilities and then deciding which capabilities to use within that set. Over time it should be possible to establish the common capabilities of the terminals in the conference using a logical ANDing process of all the capabilities received. Deciding which mode to use of the resulting set seems more problematic as a consistent notion of which mode is the best of the ones available needs to be established between all of the members in the conference. For video algorithms this may be a relatively straight forward process, but for audio this may vary depending on the application environment and personal preference. One possibility might be to employ the observation that in general, only one person will be talking at the same time. Also each RTP packet is tagged with the coding mode. By loading all the common decoders in use in the conference into system memory a terminal may be able to select the appropriate decoder as each audio packet arrives. This will require a larger memory foot print, but should not require extra processing power. To simplify this situation, new speakers may be able to implement heuristics such as using the coding algorithm of the previous speaker.

5. Use of the Feature Message

The Feature message is used for additional signalling such as transferring calls and putting people on hold. The minimum support required for this message is to identify when a message has been sent to you, and to respond with the notSupported form of the message. Much of this processing is identical to the other messages, and so this does not represent a significant burden.

This message is intended to be the mechanism by which the protocol is extended. The concept is to have optional services/protocols that bolt-on to the message. Bolt-on services have already been specified for call transfer and control of real-time streams supplied by servers.

It is expected that software modules providing services to the call control protocol will register with the feature handling components in the core layer. When a service wants to send a message, it will tell the feature handler to do so. The feature handler will keep retransmitting the feature message until it gets an ack back from the remote feature handler, or a not supported indication. The feature handler will then tell the service that the message has been sent. At the receive side, the feature handler will extract the service for who the message is intended. If a service of that type has been registered, then the message will be sent to that service and an acknowledgement will be sent. If no service of the specified name is registered, then a notSupported message will be returned. The querySupported form of the message is supported in a similar way.

Proprietary extensions to the protocol should also be made using the feature message. To ensure that proprietary extensions do not overlap with those from different vendors or future standardised messages, they should use the naming convention of:

```
     <service number>.<DNS domain name>
```

For example:

```
     myService.products.mycompany.com
```

The entire string should not exceed 255 characters in length.

6. Connecting to Stream Servers

Accessing different kinds of media within a conference in a consistent way is an important issue for call control as it simplifies the client code, but also makes the user experience more consistent. This should also apply to material introduced into a conference which is pre-stored on servers. This is especially the case if a remote user is invited to a conference who is away from their machine and has left a pre-recorded message.

Equally important is, if a server is introduced into a conference, a mechanism for making full use of the server features should be available. Extending the protocol to include a full set of media control options is not desirable, but a number of possibilities are available within the framework of SUCCESS for achieving this. The method chosen here is to make use of the feature message. Assuming that the server supports RTSP or a similar protocol, the resulting sequence of events would occur.

The user invites the server to the conference using the standard Hello message handshakes. When the server is in the conference, it sends a feature message to the user's client indicating that it supports RTSP, and what the appropriate address is.

If the user client does not have an RTSP feature registered, then the client will send back a feature notSupported message. This tells the server that RTSP control is not available. In this instance the server should proceed to play it's pre-stored material, and then exit the conference using the Bye sequence.

If the user client has registered an RTSP feature controller with the SUCCESS layer, then the client will acknowledge the feature message and pass the contents of the incoming RTSP feature message to the RTSP control mechanism. This event could be used to launch the display of a set of VCR style control buttons on the user display. The client would connect to the server specified address and issue appropriate server control messages such as HELLO and PLAY_RANGE. When the client had finished with the server it would send the GOODBYE message. At this point the server should send the SUCCESS Bye message indicating that it is leaving the conference.

Using this strategy facilitates a consistent user experience, but also allows the maximum flexibility of the invited streams to be exploited.

7. Message Encoding

SD is described using a limited set of tokens which are intended not to be extensible. Hence, its impossible to describe SUCCESS as a set of extensions to SD as is perhaps desirable.

Although the protocol described above is orthogonal to the underlying message transportation mechanism, some thoughts on message encoding are perhaps justified. An important consideration is that the message set should be extensible over time, with older terminals simply ignoring fields they do not understand. As new message elements are introduced they will likely contain multiple associated pieces of information. An efficient way of grouping these is important so that an entire message element can be ignored if required. Hence some concept of structure in messages is required..

ASN.1 is now the method of choice for encoding messages in ITU standards. The benefits of ASN.1 is that it describes messages in a powerful expressive high level way. It is similar to writing code in Pascal or C as opposed to Assembler. The downside is that it typically requires the use of a compiler to compile the messages into a rather esoteric line format.

The IETF community have a preference for encoding messages in ASCII (or equivalent). This is partly because it easily solves the problem of data representation when moving from machine to machine (ASN.1 also does this), and because it allows the data to be generated and read by humans.

Observing that it is unlikely that the ITU will want to depart from using ASN.1 and the IETF would still like to maintain a line format which is ASCII based, and there is mutual benefit from the two bodies defining standards that (if not the same) are interworkable, this section describes a mechanism for compiling ASN.1 messages into ASCII text.

The benefits of this would be that a common method of expressing high level messages could be adopted, and interworking between the standards would be trivial.

The first requirement to providing a simplified ASCII encoding is to select a sub-set of the total ASN.1 capabilities. To this end, the following keywords have been selected. All other keywords are ignored.

```
     INTEGER        OCTET       IA5String    BMPString
      STRING        SIZE       SEQUENCE OF     SET OF
     SEQUENCE      CHOICE        BOOLEAN        NULL
     OPTIONAL
```

Subset of ASN.1 keywords

To demonstrate the scheme an example is given.

A typical definition for an (complicated) ASN.1 message may look as follows:

```
startup     ::=  SEQUENCE
{
    sequence_no  INTEGER( 1..65536 ),
    name         IA5String( SIZE( 0..128 ) ),
    gUID         OCTET STRING ( SIZE( 16 ) ),
    activated    BOOLEAN
    modes        SEQUENCE
                 {
                 highmodeBOOLEAN,
                 lowmode BOOLEAN,
                 ...
                 },
    response     CHOICE
                 {
                 acknowledge  NULL,     -- NULL indicates no further data
                 silent       NULL,
                 informGroup  INTEGER( 0..65536 ),  -- Address to send
                                                    --group response to
                 ...
                 },
    id           INTEGER( 1..256 ) OPTIONAL,
    node_alerts  SEQUENCE OF INTEGER( 0..65535 ),
    complex      SEQUENCE OF SEQUENCE
                 {
                 admin_node   INTEGER( 0..256 ),
                 user_id      INTEGER( 0..256 ),
                 mode         SEQUENCE
                    {
                    video     BOOLEAN,
                    audio     BOOLEAN,
                    data      BOOLEAN,
                    ...
                    } OPTIONAL,
                 ...
```

```
                  }
    ...
}
```

From this it can be seen that there are some basic types including: INTEGER, IA5String, OCTET STRING and BOOLEAN.

There are also two `complex' structures, these being SEQUENCE and CHOICE. A SEQUENCE is similar to a structure (struct) in C and a CHOICE is similar to a union (however, the chosen option in the CHOICE is also recorded, which is not the case for a C union).

A final consideration is that you can have a SEQUENCE OF or SET OF the above types, and elements can be OPTIONAL.

In a SEQUENCE OF or SET OF construct there can be more than one of the specified component. The number of items may be constrained or unconstrained.

Elements that are marked OPTIONAL can be absent in a correctly formed message. All other elements must be present for the message to be valid.

Now lets consider how these can be represented in Unicode.

The basic mechanism is to encode all items as:

```
<name of item> <optional white space>  = <optional white
space> <value> <white space>
```

By doing this consistently, parsers can skip fields they don't understand.

Therefore the INTEGER can simply be represented as a printable string of the number, (the range of the number is not so important to the line format when represented in this way. However, the number range is probably important to the application.) e.g.

```
sequence_no = 125
```

An IA5String can be represented as a string in quotes, e.g.

```
name = "Pete"
```

... the usual back slash escapes can also be included.

OCTET STRINGs are represented using the following representation:

```
gUID = x0f1b6c0d
```

...here, each OCTET is coded as two hexadecimal digits. The leading x indicates that this is in OCTET representation.

Booleans can be coded simply as TRUE and FALSE, as in:

```
activated = TRUE
```

The SEQUENCE can be coded by including the elements of the sequence in brackets (), for example:

```
modes = ( highmode = TRUE lowmode = FALSE )
```

...doing this allows the complete sequence to be skipped if the parameter is not understood, or it is of no interest. This is important for backwards compatibility. (N.B. the ellipsis are important for coding messages using the ITU method, but they have no significance for this coding method. However, to ensure compatibility, they should be included in the message definition where appropriate.) Also note that the complete message is itself a SEQUENCE. This explains the example of the complete message shown below.

A CHOICE can be encoded using a similar scheme to the SEQUENCE, as in:

```
response = ( acknowledge = NULL )
```

... or:

```
response = ( informGroup = 137 )
```

... many choice options map to NULL, (an example of which is shown above) which is inefficient in terms of characters sent and tedious to write by hand. Conversely, this presents little problem to a program scanning and generating the text as it consistently maintains the X=Y format. However, on the whole a shorthand notation for the above of:

```
response = ( acknowledge )
```

... seems preferable. Note that the brackets are still important as this highlights that acknowledge comes from a CHOICE statement. An implementation should recognise both formats.

The OPTIONAL items is either present or not present. Unfortunately an example makes no sense.

When multiple items of the same type are included in a message using the SEQUENCE OF or SET OF encoding, this can be done simply by including the item multiple times, as in:

```
node_alerts = 0 node_alerts = 5000 node_alerts = 12
```

... this is quite wasteful in terms of characters, and so the following compacted encoding could be used:

```
node_alerts = 0 = 5000 = 12
```

... the rule that allows this is that if you get the = token when you expected to receive an item name, you should use the most recently collected item name, subject to the level of parenthesis.

As a final, complicated example, the `complex' component shown above can be encoded as:

```
complex = (    admin_node = 20
        user_id = 6
        mode = ( video = TRUE audio = TRUE data = FALSE )
    )

    = ( admin_node = 5
        user_id = 5
    )
```

To sum up, a complete example of the message would be:

```
startup = (
    sequence_no = 125
    name = "Pete"
    gUID = x0f1b6c0d
    activated = TRUE
    modes = ( highmode = TRUE lowmode = FALSE )
    response = ( informGroup = 137 )
    id = 12
    node_alerts = 0 = 5000 = 12
    complex = (
        admin_node = 20
        user_id = 6
        mode = ( video = TRUE audio = TRUE data = FALSE )
        )
        = (
        admin_node = 5
```

```
            user_id = 5
            )
        )
```

Note that because each element is tagged, there order is not important. Therefore, the above message could equally be represented as:

```
startup = (
    name = "Pete"
    activated = TRUE
    node_alerts = 0
    sequence_no = 125
    modes = ( highmode = TRUE lowmode = FALSE )
    id = 12
    node_alerts = 5000 = 12
    complex = (
        admin_node = 20
        user_id = 6
        mode = ( video = TRUE audio = TRUE data = FALSE )
        )
    response = ( informGroup = 137 )
    complex = (
        admin_node = 5
        user_id = 5
        )
    gUID = x0f1b6c0d     // We can have comments too
    )
```

A final comment is that message definitions rarely map directly to the base (INTEGER, OCTET) types. I.e. the definition above might be encoded as:

```
startup ::=  SEQUENCE
{
    sequence_no     Seq_no,
    name            IA5String(SIZE(0..128)),
    gUID            conference_ID,
    activated       BOOLEAN,
    modes           Modes,
        .
        .
        .
```

elsewhere the following definitions would appear:

```
Seq_no               INTEGER(1..65536),
conference_ID  ::=  OCTET STRING (SIZE(16)),
Modes          ::=  SEQUENCE
               {
               highmode  BOOLEAN,
               lowmode   BOOLEAN,
                              ...
                              }
```

This is a better way to do the message definition, for all the reasons that you would do the same in any piece of software. The coding method is not affected by this as it is a process of macro expansion to get to the message definition we started with.

8. Address of Author

Peter Cordell
BT Labs
MLB 4/40
Martlesham Heath
Ipswich IP5 7RE
e-mail: pete.cordell@bt-sys.bt.co.uk

References

This document has drawn heavily on the following sources:

SDP M. Handley, V. Jacobson "SDP: Session Description Protocol" Internet Draft, draft-ietf-mmusic-sdp-02.txt, Work in Progress, Feb 1996.

SIP M. Handley, E. Schooler "SIP: Session Invitation Protocol" Internet draft, draft-ietf-mmusic-sip-01.txt, June 1996

SCIP H. Schulzrinne, "Simple Conference Invitation Protocol", Internet draft, draft-ietf-mmusic-scip-00.txt, Feb 1996

H.245 ITU-T, "Control Protocol for Multimedia Communication" Nov 1995

H.225 ITU-T, "Media Stream Packetisation and Synchronisation on Non-Guarenteed Quality of Service LANs", May 1996

X.680 ITU-T, "Abstract Syntax Notation One (ASN.1): Specification of Basic Notation", July 1994

To Do
Add scenarios.
Check centralised control such as used in call centres.
Write text on how the e164 address field should be used.
Re-visit how properties of streams are transmitted.
Re-visit gateway connectivity now that new pseudo-code has been put in.