Telecommunications Standardisation sector
Study Group 15
Experts Group for Very Low Bitrate Video Telephony
Experts' Group for Video Coding and Systems in
ATM and Other Network Environments

LBC-95-296 / AVC-829
Version 1
17-20 October, Darmstadt
24-27 October, Yokosuka

Source: BT
Title: Bi-directional logical channel signalling
Purpose: Proposal

This contribution reviews the current definition of the bi-directional logical channel signalling protocol in H.245 and finds that it is unnecessarily complex and incurs unnecessary delay.

A simplified protocol is proposed that unifies the protocol used for uni-directional and bi-directional logical channel signalling.

Problems with the current specification of logical channel signalling in H.245

The uni-directional logical channel signalling protocol in H.245 is very simple: one terminal sends a message to request opening a channel, and the far-end terminal responds with a positive or negative acknowledgement.

The bi-directional logical channel signalling protocol in H.245 is not so simple. Although it may be an elegant solution that uses two LCSEs at each end, it is quite complicated, and probably incurs more round trip delays than the absolute minimum. In addition, the compound state transition tables are quite difficult to read.

The requirements of the logical channel signalling protocols in H.245

Bi-directional signalling has been defined to be more complex than uni-directional signalling to avoid the 'crossing-in-the-mail' problem when two bi-directional channels would be set up when only one was really wanted. However, a similar problem exists in uni-directional signalling: consider the following example.

> Both terminals are capable of a number of different audio algorithms, G.711, G.723, G.728, but can only receive and transmit with the same algorithm. The capability sets enable this knowledge to be shared by the two end terminals. But which one opens the first audio logical channel, and which algorithm is chosen? One could choose to open a channel for G.723, while the other could choose to open a channel for G.728, and the open messages could 'cross in the post'.

There are two similar problems that the signalling protocol must be designed to overcome.

- Both terminals may initiate requests for uni- or bi-directional logical channels at about the same time, these 'cross in the post', but known capability descriptions indicate that both requests can not be satisfied at the same time.

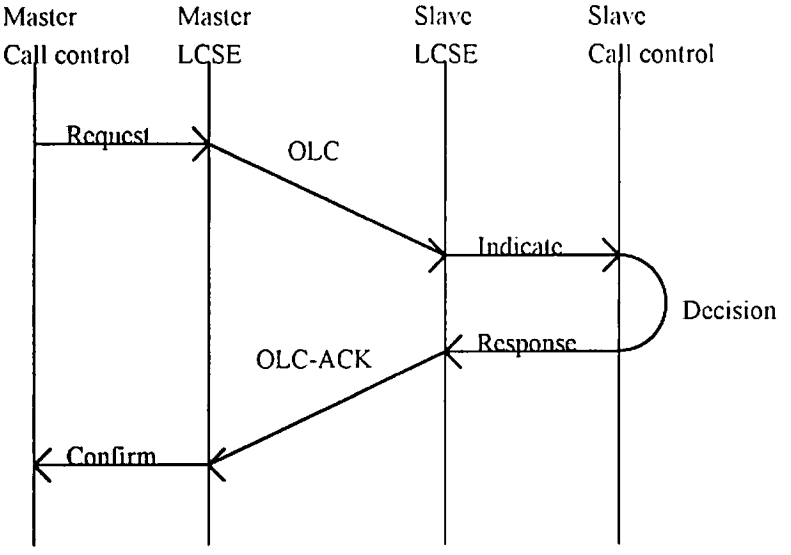
- Both terminals may initiate requests for bi-directional logical channels at about the same time, these 'cross in the post', and known capability descriptions indicate that both requests can be satisfied at the same time, but neither terminal knows whether two channels are needed or only one.

The first of these can happen with both uni- or bi-directional logical channels, an example being the situation given above, and the best solution is a protocol that accepts one of the requests and rejects the other. The second only happens with bi-directional logical channels, for example when both terminals have capability for multiple simultaneous video streams with H.223 Adaptation Layer 3. As there is no simple way to indicate 'want', the best solution is a protocol that assumes conflict rather than desire for two, and accepts one of the requests and rejects the other. If one of the terminals did in fact want two bi-directional channels, it can make another request.

Outline of a simplified protocol

As conflicting requests come from both uni-directional and bi-directional channel requests, it should be possible to use the same protocol for opening both uni-directional and bi-directional channels.

Figure 1 shows the protocol when there is no conflict. A request from call control leads to an OpenLogicalChannel PDU. This is indicated to the remote call control which makes a decision (accept) and responds with OpenLogicalChannelAck, which, when received at the initiating terminal, confirms the logical channel establishment to the call control.



OLC: OpenLogicalChannel OLC-ACK: OpenLogicalChannelAck

Figure 1. Logical channel set up when there are no conflicting requests.

Provided that one terminal is master and the other is slave, and both are fully aware of their responsibilities, and of each other's declared capabilities, and of requests that have been sent and received, and if the slave *trusts* the master, then only one message needs to be sent in each direction to open either a uni-directional or bi-directional channel.

The terminal that wants to open a channel sends OpenLogicalChannel, with logicalChannelNumber and forwardLogicalChannelParameters for a uni-directional channel, and with reverseLogicalChannelParameters as well for a bi-directional channel.

If the request is from the slave to the master, and the master recognises a conflict with a request it made earlier, either on grounds of insufficient capability to satisfy both requests or even if there is sufficient capability, it judges the request to be similar to its earlier one and possibly an undesired duplicate, *it shall reject it immediately.*

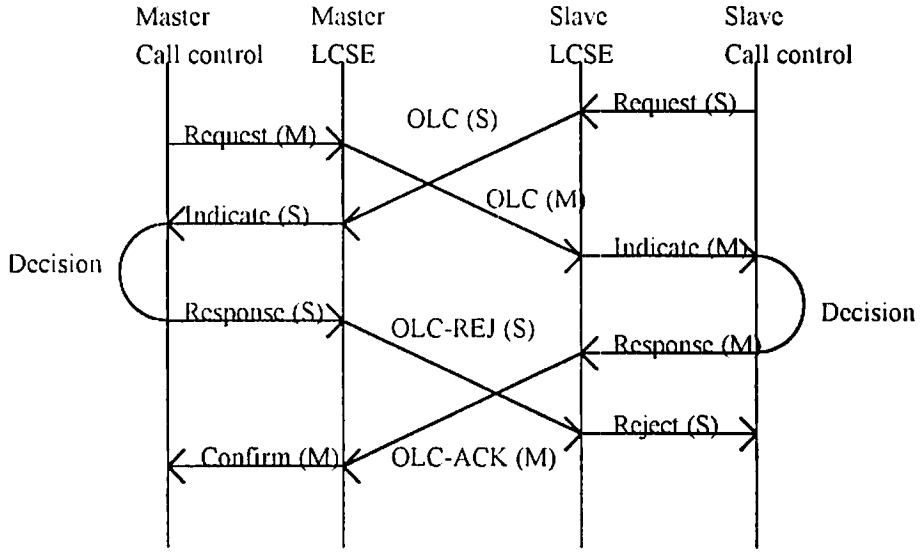
If the request is from the master to the slave, the slave shall respond immediately, making its decision to accept or reject the request according to which logical channels have already been set up, and not what it has already requested. Although the slave may recognise a conflict with a request it made earlier, either on grounds of insufficient capability to satisfy both requests or even if there is sufficient capability, it judges the request to be similar to its earlier one and possibly an undesired duplicate, *it shall ignore this, knowing that the master has come to the same conclusion and will reject it.*

Note. A fully compliant master terminal will not surprise the slave terminal by unexpectedly accepting a conflicting request. Clearly a non-compliant master may do so, but in this case the slave can recover the situation

by immediately closing the logical channels that it initiated opening and which the master should never have accepted.

Comment. This protocol gives a slightly unfair advantage to the master terminal, as in the case of conflict, the master's request succeeds while the slave's fails. It is therefore beneficial to prevent terminals of equal status (i.e. excluding MCUs) being able to fix the result of the master slave determination procedure in their favour. Annex 4 provides a means to ensure that fixing of the result is not possible.

Figure 2 shows the protocol when there is conflict. Request from both call control entities leads to OpenLogicalChannel PDUs that cross 'in the post'. Each of these is indicated to the remote call control entities which make decisions: the master recognises the conflict and rejects; while the slave, which may also recognise the conflict, accepts. The master responds with OpenLogicalChannelReject and the slave with OpenLogicalChannelAck. Confirmation is passed to the master's call control entity and rejection to the slave's. The slave may repeat its attempt to establish a channel if both were really desired, and this attempt would succeed - this is not shown in figure 2 but the behaviour would be like that shown in figure 1.



OLC: OpenLogicalChannel OLC-ACK: OpenLogicalChannelAck

OLC-REJ: OpenLogicalChannelReject (M): Initiated by master (S): Initiated by slave

Figure 2. Logical channel set up when there are conflicting requests.

Specification of the simplified protocol

This section specifies the changes that would be needed to H.245 to adopt the simplified protocol in place of the current logical channel protocols.

Open Bi-directional Channel Request (OBSCE)

This protocol is currently used by the slave to request the master to open bi-directional logical channels. It is independent of the protocol that is used to open bi-directional logical channels. This protocol is no longer needed.

The syntax and semantics for OpenBiDirectionalChannelRequest, OpenBiDirectionalChannelAck, OpenBiDirectionalChannelReject, and OpenBiDirectionalChannelRelease should be deleted.

The open bi-directional channel procedures defined in section 8.7 should also be deleted.

Logical channel signalling messages

The syntax for these messages should be changed as shown in Annex 1. Forward logical channel parameters are retained as before, but reverse logical parameters are changed, as the port number and H.222 parameters are not known at the initiating end. The parameters, together with associated logical channel number, are moved to the acknowledgement message, where they be appropriately set.

The OpenLogicalChannelAck message is adapted for bi-directional requests to allow it to contain parameters concerning the reverse logical channel.

The semantics for these messages should be changed as shown in Annex 2.

Bi-directional Logical Channel Signalling Procedures (B-LSCE)

This protocol is currently used to open bi-directional logical channels. It is defined in terms of the uni-directional logical channel signalling procedures (LCSE), making use of an out-going and in-coming LCSE at each end.

This protocol is no longer needed, and the text defining it in section 8.5 should be deleted.

Logical Channel Signalling Procedures (LSCE)

This protocol is currently used to open uni-directional logical channels. It requires some small modifications to make it applicable for opening bi-directional logical channels using the same protocol.

The changes in the parameters of OpenLogicalChannel and OpenLogicalChannelAck require changes to tables 25, 26, 27, 28, and 29, and figure 12, and the text in 8.4.2.3 describing the parameters.

No change to the protocol or the SDLs would be needed.

Annex 3 contains new text for the introduction section to the protocol definition.

Note. This protocol allows a bi-directional logical channel to be closed by the initiating terminal using a single CloseLogicalChannel message.

Terminology definitions

The current definitions section of H.245 defines logical channels and bi-directional logical channels but not uni-directional logical channels. These definitions should be updated to the following.

Logical Channel: A logical channel is a uni-directional or bi-directional path for the transmission of information.

Uni-directional Logical Channel: A uni-directional logical channel is a path for the transmission of a single elementary stream from one terminal to another.

Bi-directional Logical Channel: A bi-directional logical channel is a pair of associated uni-directional logical channels, one in each direction of transmission.

Conclusion

This contribution has stated that the current definition of the bi-directional logical channel signalling protocol in H.245 is unnecessarily complex and incurs unnecessary delay.

A simplified protocol has been defined that unifies the protocol used for uni-directional and bi-directional logical channel signalling.

END

Annex 1
Proposed Syntax for Logical Channel Signalling

This annex defines the syntax that would be needed for logical channel signalling in the simplified protocol. Only the syntax that is changed is listed. As an editorial improvement, logicalChannelNumber has been changed to forwardLogicalChannelNumber and reverseLogicalChannelNumber as appropriate.

```
================================================================================
-- Logical channel signalling definitions
================================================================================

-- 'Forward' is used to refer to transmission in the direction from the terminal making the
-- original request for a logical channel to the other terminal, and 'reverse' is used to refer
-- to the opposite direction of transmission, in the case of a bi-directional channel request.

OpenLogicalChannel                  ::=SEQUENCE
{
    forwardLogicalChannelNumber     LogicalChannelNumber,

    forwardLogicalChannelParameters SEQUENCE
    {
        portNumber                  INTEGER (0..65535),
        dataType                    DataType,
        multiplexParameters         CHOICE
        {
            h222LogicalChannelParameters H222LogicalChannelParameters,
            h223LogicalChannelParameters H223LogicalChannelParameters,
            ...
        },
        ...
    },

    -- Used to specify the reverse channel for bi-directional open request

    reverseLogicalChannelParameters SEQUENCE
    {
        dataType                    DataType,
        multiplexParameters         CHOICE
        {
            -- H.222 parameters are never present in reverse direction
            h223LogicalChannelParameters H223LogicalChannelParameters,
            ...
        } OPTIONAL,                 -- Not present for H.222
        ...
    } OPTIONAL,                     -- Not present for uni-directional channel request

    ...
}

OpenLogicalChannelAck               ::=SEQUENCE
{
    forwardLogicalChannelNumber     LogicalChannelNumber,

    reverseLogicalChannelParameters SEQUENCE
    {
        reverseLogicalChannelNumber LogicalChannelNumber,
```

```
        portNumber                          INTEGER (0..65535),
        multiplexParameters                 CHOICE
        {
            h222LogicalChannelParameters H222LogicalChannelParameters,
            -- H.223 parameters are never present in reverse direction
            ...
        } OPTIONAL,                         -- Not present for H.223
        ...

    } OPTIONAL,                             -- Not present for uni-directional channel request

    ...
}

OpenLogicalChannelReject                    ::=SEQUENCE
{
    forwardLogicalChannelNumber             LogicalChannelNumber,
    cause                                   CHOICE
    {
        unspecified                         NULL,
        dataTypeNotSupported                NULL,
        dataTypeNotAvailable                NULL,
        unknownDataType                     NULL,
        dataTypeALCombinationNotSupported   NULL,
        ...
    },
    ...
}
```

Annex 2
Proposed Semantics for Logical Channel Signalling

This annex defines the semantics that would be needed for logical channel signalling in the simplified protocol. Only the semantics that are changed are listed. As an editorial improvement, logicalChannelNumber has been changed to forwardLogicalChannelNumber and reverseLogicalChannelNumber as appropriate.

7.3 Logical channel signalling messages

This set of messages is for logical channel signalling. The same set of messages is used for uni-directional and bi-directional logical channel signalling; however, some parameters are only present in the case of bi-directional logical channel signalling.

'Forward' is used to refer to transmission in the direction from the terminal making the original request for a logical channel to the other terminal, and 'reverse' is used to refer to the opposite direction of transmission, in the case of a bi-directional channel request.

7.3.1 Open Logical Channel

This is used to attempt to open a logical channel connection between an out-going LCSE and a peer in-coming LCSE.

forwardLogicalChannelNumber indicates the logical channel number of the forward logical channel that is to be opened.

forwardLogicalChannelParameters: include parameters associated with the logical channel in the case of attempting to open a uni-directional channel and parameters associated with the forward logical channel in the case of attempting to open a bi-directional channel.

portNumber is a user to user parameter that may be used by a user for such purposes as associating an input or output port, or higher layer channel number, with the forward logical channel.

dataType indicates the data that is to be carried on the forward logical channel.

multiplexParameters indicate parameters specific to the multiplex, H.222 or H.223, that is used to transport the forward logical channel.

reverseLogicalChannelParameters: include parameters associated with the reverse logical channel in the case of attempting to open a bi-directional channel. Its presence indicates that the request is for a bi-directional logical channel with the stated parameters, and its absence indicates that the request is for a uni-directional logical channel.

Note. H.222 parameters are not included in reverseLogicalChannelParameters as their values are not known to the terminal initiating the request.

7.3.2 Open Logical Channel Acknowledge

This is used to confirm acceptance of the logical channel connection request from the peer LCSE. In the case of a request for a uni-directional logical channel, it indicates acceptance of that uni-directional logical channel. In the case of a request for a bi-directional logical channel, it indicates acceptance of that bi-directional logical channel, and indicates the appropriate parameters of the reverse logical channel.

forwardLogicalChannelNumber indicates the logical channel number of the forward logical channel that is being opened.

reverseLogicalChannelParameters is present if and only if responding to a bi-directional channel request.

reverseLogicalChannelNumber indicates the logical channel number of the reverse logical channel.

portNumber is a user to user parameter that may be used by a user for such purposes as associating an input or output port, or higher layer channel number, with the reverse logical channel.

multiplexParameters indicate parameters specific to the multiplex, H.222 or H.223, that is used to transport the reverse logical channel.

Note. H.223 parameters are not included in reverseLogicalChannelParameters as their values were specified in the OpenLogicalChannel request message.

7.3.3 Open Logical Channel Reject

This is used to reject the logical channel connection request from the peer LCSE.

Note. In the case of a bi-directional request, rejection applies to both forward and reverse channels. It is not possible to accept one and reject the other.

forwardLogicalChannelNumber indicates the logical channel number of the forward channel specified in the request that is being rejected.

An additional codepoint has been suggested:

dataTypeALCombinationNotSupported: the terminal was not capable of supporting the dataType indicated in OpenLogicalChannel with the Adaptation Layer type indicated in H223LogicalChannelParameters.

7.3.4 Close Logical Channel

This is used by the out-going LCSE to close a logical channel connection between two peer LCSEs.

Note. In the case of a bi-directional logical channel, this closes both forward and reverse channels. It is not possible to accept one and reject the other.

forwardLogicalChannelNumber indicates the logical channel number of the forward channel of the logical channel that is to be closed.

Table 7 remains unchanged.

7.3.5 Close Logical Channel Acknowledge

This is used to confirm the closing of a logical channel connection.

forwardLogicalChannelNumber indicates the logical channel number of the forward channel of the logical channel that is to be closed.

Annex 3
Proposed Introduction for Logical Channel Signalling Protocol

This annex provides new text for the introduction to the section that defines the logical channel signalling protocol. This text would replace that currently in section 8.4.1

The protocol specified here provides reliable opening and closing of uni-directional and bi-directional logical channels using acknowledgement procedures.

It is referred to as the Logical Channel Signalling Entity (LCSE). Procedures are specified in terms of primitives at the interface between the LCSE and the LCSE user, and LCSE states. Protocol information is transferred to the peer LCSE via relevant messages defined in section 6. There is an out-going LCSE and an in-coming LCSE. At each of the out-going and in-coming sides there is one instance of the LCSE for each logical channel. LCSE error conditions are reported.

Either terminal may initiate the opening of a logical channel by issuing the ESTABLISH.request primitive to its LCSE. Successful opening is indicated by the ESTABLISH.confirm primitive and failure by the RELEASE.confirm primitive.

A terminal is made aware of a request to open a logical channel by receipt of the ESTABLISH.indication primitive. A terminal accepts the request by issuing the ESTABLISH.response primitive and rejects it by issuing the RELEASE.request primitive.

The logical channel shall only be used for audiovisual and data communication while in the ESTABLISHED state.

Note. Some recommendations that use this Recommendation may define some default logical channels. These shall be considered ESTABLISHED from the start of communication and so do not need to be opened using these procedures. They may, however, be closed by these procedures, and subsequently be re-opened for the same or a different purpose.

On occasions there may be conflicting requests for logical channels, caused by both terminals initiating requests at about the same time. It may be possible to determine that there is conflict from knowledge of exchanged capabilities. On other occasions, when opening bi-directional logical channels, both terminals may initiate the opening of a channel for the same purpose, even though the exact parameters requested may be different, and both terminals have sufficient capability for both requests. Terminals shall be capable of detecting when both of these situations have arisen, and shall act as follows.

Before logical channels can be opened, one terminal must be determined as the master terminal, and the other as the slave. The protocol defined in 8.2 provides one means to make this decision. The master terminal shall reject immediately any request from the slave that it identifies as a conflicting request. The slave terminal may identify such conflicts, but shall ignore the conflict for the purposes of responding to the requests of the master.

Note. In the second type of conflict defined above, it is impossible to distinguish when two bi-directional channels are actually wanted from the case when only one is wanted. Terminals shall respond assuming that only one is wanted, but a terminal may subsequently repeat its request if the assumption was incorrect.

The following text provides an overview of the operation of the protocol. In the case of any discrepancy with the formal specification of the protocol that follows, the formal specification will supersede.

Either terminal may initiate the opening of a logical channel by transmitting OpenLogicalChannel, indicating the data that will be conveyed on the logical channel. Transmission on the logical channel shall not start until OpenLogicalChannelAck has been received.

A terminal may change the data type that is transmitted on a logical channel by sending another OpenLogicalChannel, without first having to send CloseLogicalChannel. But no data can be transmitted between this time and when OpenLogicalChannelAck is received, as the LCSE is no longer in the ESTABLISHED state.

A logical channel may be closed by sending CloseLogicalChannel. The transmission of data on the logical channel shall stop before CloseLogicalChannel is sent.

On receiving OpenLogicalChannel, a terminal shall send OpenLogicalChannelAck if it is able to satisfy the request, and OpenLogicalChannelReject if it is unable to satisfy the request. In the case of bi-directional logical channels, OpenLogicalChannelAck will contain information about the reverse channel, including the logical channel number.

Mode switching should be performed either by closing and opening different logical channels, or by changing the use of a particular logical channel as above.

A terminal that is no longer capability of processing the signals on a logical channel should take appropriate action: this should include closing the logical channel and transmitting the relevant (changed) capability information to the remote terminal.

Annex 4
A fair master slave determination procedure

This annex specifies a way to ensure that the result of the master slave determination is fair. Currently it is possible for terminals to choose big numbers to try to become the master. If two connected terminals are doing this, the range of numbers they may be selecting from may be quite small, and choosing the same number may not be as unlikely as it should be.

The following example is given to illustrate the recommended concept. Consider choosing between three numbers, 1, 2, 3, and having the following results, '2 beats 1', '3 beats 2' and '1 beats 3'. This simple scheme prevents cheating by providing no gain by choosing one number ahead of any other.

The concept can easily be extended to larger ranges of numbers by using modulo arithmetic. Considering eight bit numbers, if the result, modulo 256, of taking one number away from the other is less than 128 one number 'wins' and if more than 128 the other number 'wins'. Examples are '64 beats 32', '192 beats 160', '32 beats 240', '126 beats 255'. Draws are achieved when the difference is 0 or 128 modulo 256.

Comparing two numbers in this way using modulo arithmetic is negligibly more complicated than simply comparing on size. As it prevents cheating, it should be adopted.

The only disadvantage of banning cheating is that in some cases we may want to enforce cheating: we may want MCUs to be able to choose whether they are masters or slaves. To allow this in H.324, we have split the range of numbers into three, and have allowed terminals only to choose from the middle range. This approach does not work with the scheme described above.

To keep this functionality, but to avoid terminal-terminal cheating, we propose splitting the 32 bit status determination number into two components, a terminal type (of say 8 bits) and a random number (of say 24 bits). The rule would then be: if the terminal type numbers are different, the one with the larger number becomes master, but if they are the same, the random numbers are compared using the modulo arithmetic described above.

By defining all H.324 terminals to have the same terminal type, for example 128, we retain the possibility of using different terminal type numbers for MCUs and enable them to decide to be master or slave. It could also be extended to allow other terminal types, interworking units, servers etc., to have different terminal type numbers and 'fix' the result, even when competing with MCUs.