Connection Management Access Protocol (CMAP) Specification

John DeHart   Mike Gaddis   Rick Bubenik

Version 2.1.1
May 7, 1992

Applied Research Laboratory Working Note —**ARL-89-06**
Department of Computer Science Technical Report —**WUCS-92-01**

Applied Research Laboratory
Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
St. Louis, Missouri 63130-4899
Telephone: 314-935-6160
FAX: 314-935-7302
Email: {jdd, gaddis, rick}@arl.wustl.edu

ABSTRACT

This document specifies a *Connection Management Access Protocol* (CMAP) for managing multipoint connections in high-speed packet switched networks. We target CMAP to networks employing the *Asynchronous Transfer Mode* (ATM) communication standard. We define a *multipoint connection* as a communication channel between two or more *clients* of the network, where all data sent by one client is received by all other clients who have elected to receive. A *point-to-point connection* is a special case of a multipoint connection involving only two clients. CMAP specifies the access procedures exercised by clients to create, modify and delete multipoint connections. Once a connection is established, clients exchange data using protocols that are specified separately from CMAP. To establish a multipoint connection, a client first creates a *call* between itself and the network. The client creating a *call* is designated the *owner* of the *call*. Additional endpoints are added either by invitation from the owner, invitation from another client of the network, or by explicitly requesting to be added. These three modes are sufficient for supporting point-to-point communication (for example, a telephone *call*), many-to-many communication (for example, a conference *call* or data exchange), one-to-many communication (for example, broadcast video), and many-to-one communication (for example, distributed data collection).

Version 1.0: Initial document containing CMAP message formats and little else.
Version 2.0: Added introductory sections on ATM networks, the call model and the protocol stack (these sections were derived mostly from papers published by the authors). Changed many of the CMAP message formats and added some new messages. Added detailed explanations where deemed necessary to highlight obscure CMAP features. Added implementation and future direction sections (incomplete).
Version 2.1: Major editing rewrite of Sections 1 through 8.5 from version 2.0 (now organized as Sections 1 through 6). More editing to come.
Version 2.1.1: Minor editing changes from version 2.1. More editing to come (temporarily on hold).

Copyright Notification

Authors:
John DeHart, Mike Gaddis and Rick Bubenik

Table of Contents

Author: **John DeHart Mike Gaddis Rick Bubenik**
Organization: **Applied Research Laboratory**

Project: **Zeus Project**
File: **/BPN.SW/documentation/CM/CMAP.spec.rick**
Created: **February 26, 1992**
Modified: **May 7, 1992 5:12 pm by Gaddis**

Acronym List

The following acronyms are used within this document in reference to ATM networks, fast packet switches, and our protocols.

ACK — Acknowledgment

ATM — Asynchronous Transfer Mode

BPN — Broadcast Packet Network

BISDN — Broadband Integrated Services Digital Network

BRDP — BPN Reliable Datagram Protocol

BTC — Broadcast Translation Circuit

BTP — BPN Transport Protocol

BTT — Broadcast Translation Table

BW — Bandwidth

CCITT — International Telegraph and Telephone Consultative Committee

CLP — Cell Loss Priority (ATM header field)

CM — Call Management

CMAP — Connection Management Access Protocol

CMNP — Connection Management Network Protocol

CN — Copy Network

COM — Commit

CON — Confirmation

CP — Control Processor

CTL — CMAP Transport Layer

GFC — Generic Flow Control (ATM header field)

HEC — Header Error Check (ATM header field)

ISO-OSI — International Standards Organization - Open System Interconnection

PDU — Protocol Data Unit

PT — Payload Type (ATM header field)

QOS — Quality of Service

NACK — Negative acknowledgment

NNI — Network Node Interface

REQ — Request

RES — Response

RN — Routing Network

SF — Switch Fabric

SONET — Synchronous Optical NETwork

SM — Switch Module

SMI — Switch Module Interface

UNI — User Network Interface

VC — Virtual Channel

VCI — Virtual Channel Identifier (ATM header field)

VCXT — Virtual Channel Translation Table

VP — Virtual Path

VPI — Virtual Path Identifier (ATM header field)

VPXT — Virtual Path Translation Table

Connection Management Access Protocol (CMAP) Specification

John DeHart   Mike Gaddis   Rick Bubenik

Version 2.1.1
May 7, 1992

Applied Research Laboratory Working Note —**ARL-89-06**
Department of Computer Science Technical Report —**WUCS-92-01**

Applied Research Laboratory
Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
St. Louis, Missouri 63130-4899
Telephone: 314-935-6160
FAX: 314-935-7302
Email: {jdd, gaddis, rick}@arl.wustl.edu

ABSTRACT

This document specifies a Connection Management Access Protocol (CMAP) for managing multipoint connections in high-speed packet switched networks. We target CMAP to networks employing the Asynchronous Transfer Mode (ATM) communication standard. We define a multipoint connection as a communication channel between two or more clients of the network, where all data sent by one client is received by all other clients. A point-to-point connection is a special case of a multipoint connection involving only two clients. CMAP specifies the access procedures exercised by clients to create, modify and delete multipoint connections. Once a connection is established, clients exchange data using protocols that are specified separately from CMAP. To establish a multipoint connection, a client first creates a call between itself and the network. The client creating a call is designated the owner of the call. Additional endpoints are added either by invitation from the owner, invitation from another client of the network, or by explicitly requesting to be added. These three modes are sufficient for supporting point-to-point communication (for example, a telephone call), many-to-many communication (for example, a conference call), one-to-many communication (for example, broadcast video), and many-to-one communication (for example, distributed data collection).

1. Introduction

This document describes a *call* model for multipoint connections in a switched Asynchronous Transfer Mode (ATM) networks and specifies the Connection Management Access Protocol (CMAP) that allows network *clients* to create, manipulate and delete multipoint, multiconnection communication channels, which we term *calls*. A *multipoint call* is a call involving two or more clients; a *point-to-point* call is a special case of a multipoint *call* involving only two clients. Data sent over a connection by one participant in a call is received by all other participants electing to receive on this connection, although reliable delivery is not guaranteed by the network. Calls are allowed to change dynamically during their lifetime, in terms of the number of participants, the number of connections and the reserved bandwidth of the connections.

When a call is created, one or more connections are established between the network and the client who created the call. This client is designated the *owner* of the *call*. Additional clients, or *endpoints*, are added to the call by: 1) invitation from the owner, where the invited party has the option of refusing the invitation, 2) request from a client not currently in the call to be added, where the owner has the option of denying the request, or 3) request from a third party, not necessarily in the call, to add a client, where both the owner and the client being added have the option to refuse. Once a call has been created, additional connections can be added to the call as well. Receive/transmit permissions are present to limit the receive/transmit capability of each endpoint on each connection.

CMAP defines the interface between clients and the network used to create, manipulate and delete calls. As such, CMAP is an ATM *User Network Interface* (UNI) signaling protocol [3] [14]. It is layered over a reliable substrate, which we term CTL (for CMAP Transport Layer). We do not specify the CTL protocol. Rather, we list the requirements for CTL, which are generally met by several existing transport protocols (for example, TCP/IP over type 4 AAL). In this way, CMAP implementors can choose the most suitable CTL for their implementation environment.

Although CMAP and the associated call model provide a rich set of operations and capabilities, CMAP implementations are not required to be complete: CMAP implementations can omit certain operations and capabilities, while still adhering to the general message layouts and request/acknowledgment handshake procedures. A certain set of core capabilities are required, which we term *minimal* CMAP (Section 8). All other capabilities are optional. One example of an optional capability is support for multipoint connections. CMAP can be used on networks that only support point-to-point connections by omitting the operations related to multipoint.

Addressing is another area where CMAP remains flexible by not dictating any one addressing scheme. Rather, CMAP supports multiple adressing diciplines and multiple routing protocols. Currently defined addresses schemes include IP (Internet Protocol) addressing [18], public network E.164 addressing [13], and OSI NSAP addressing [17]. An implementation of CMAP may support any or all of these schemes, plus others.

The remainder of this document is organized as shown in Table 1.

TABLE 1. Document Layout.

Section and Title	Description
Section 1: **Introduction**	Call model and CMAP overview.
Section 2: **ATM Networks**	Introduction to switched, connection oriented ATM networks.
Section 3: **Call Model**	Detailed description of the call model.
Section 4: **Signaling the Network**	Description of CMAP layering over ATM, signaling connections, multidrop access links, surrogate signaling and CTL requirements.
Section 5: **CM Network Protocol**	Discussion of the internal network protocol that supports CMAP.
Section 6: **CMAP Concepts**	General information on CMAP applicable to all CMAP operations.
Section 7: **CMAP Operations**	Message formats and state diagrams of all CMAP operations.
Section 8: **CMAP Implementations**	Definitions of minimal and complete CMAP implementations.
Section 9: **CMAP Future Directions**	List of CMAP enhancements being considered.
Appendix A: **References**	List of related references.
Appendix B: **CMAP Field Values**	Numerical values for CMAP message parameters.
Appendix C: **CMAP Error Codes**	Numerical values for CMAP error codes.
Appendix D: **Minimal CMAP**	Tabular listing of requirements for minimal CMAP.
Appendix E: **Endpoint Mappings**	Tabular listing of all possible endpoint receive/transmit mappings.

2. Switched ATM Networks

This section presents an overview of switched network architectures, the ATM standard, ATM network connections (or *cell pipes,* as we term them) and uses of the two types of ATM connections: *Virtual Path* and *Virtual Channel*.

2.1 Network Architecture

An example ATM network is shown in Figure 1. The network consists of *clients*, *exterior nodes* (nodes that interface to clients), and *interior nodes* (nodes that interface to other nodes only), all interconnected by *fiber optic links*. A client signals the network to set up calls with other clients by sending *control* messages to exterior nodes, which carry out the requested client operations. If the request requires the services of other nodes, then control messages are exchanged between nodes (exterior and interior) in order to satisfy the request. The access protocol hides the internal topology of the network from clients (that is, clients cannot distinguish a single node network from an arbitrarily interconnected multinode network).

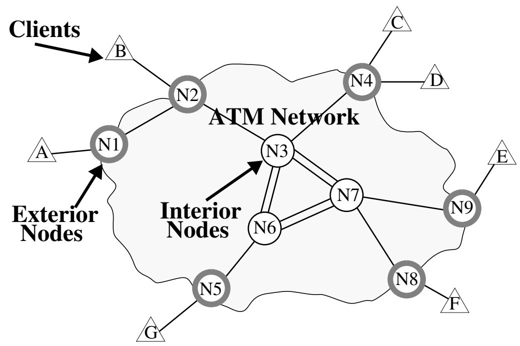


Figure 1. Example ATM Network.

Each node in the network contains one or more ATM switches [29][33][34][36][37][58][61]. The switches route each ATM cell (packet) to the desired destination link(s) based upon header fields in the cell (Section 2.2). In order to keep up with line speeds, the switches perform all routing in hardware. Since the time interval within which each cell must be routed is very small, tables in the switches are preconfigured with routing information. This makes ATM networks more suitable for connection oriented traffic, where the switch tables can be configured during connection setup. Connectionless traffic can be accommodated via *overlay* networks utilizing special purpose routers or datagram processors [5][9].

Figure 1 shows the architecture of one ATM switch, Turner's *Broadcast Packet Switch* [58][61]. This switch contains a *Copy Network* (CN) concatenated to a *Routing Network* (RN). ATM cells (packets) enter the switch on the left.

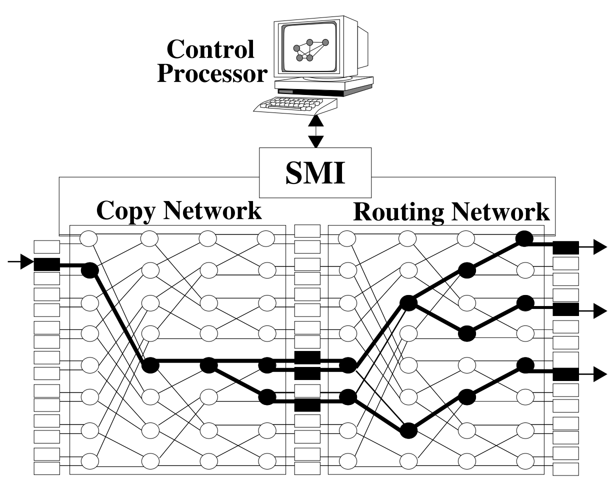


Figure 2. Architecture of Turner's Broadcast Packet Switch Demonstrating Multicast Routing.

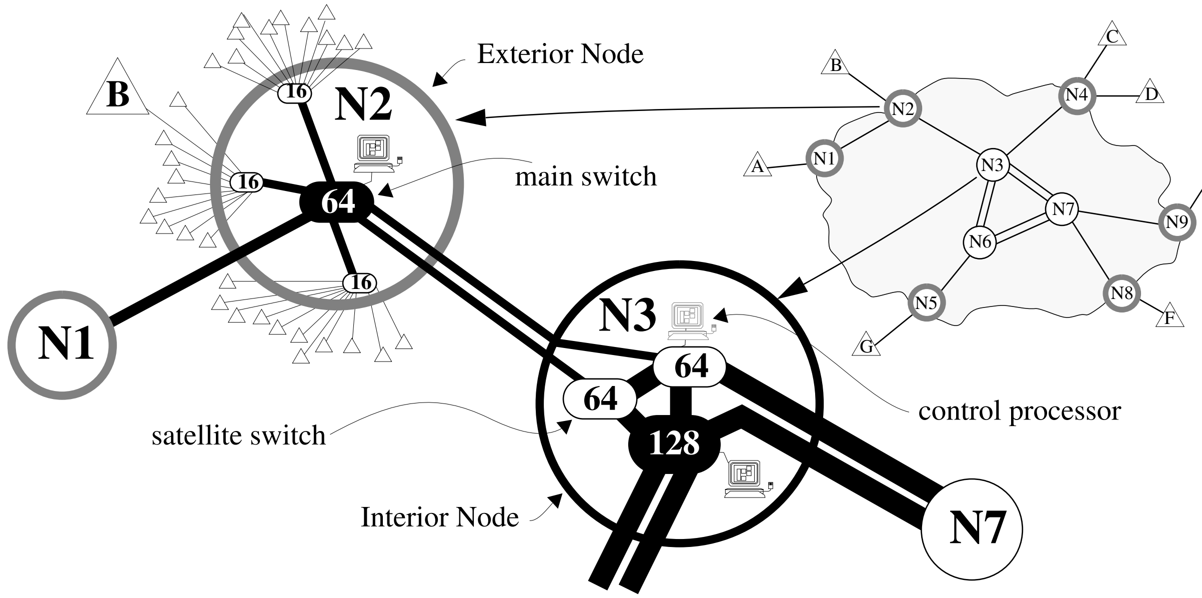


Figure 3. Generalized Node Architecture for Interior and Exterior Network Nodes.

Multicast cells, destined for several locations, are replicated by the CN, then routed to the appropriate destination by the RN. Point-to-point cells follow an arbitrary path through the CN (following the path of "least resistance"), then are routed by the RN. Cells leave the switch on the right, where they traverse fiber optic links to other switches or clients.

The switch is controlled by the *Control Processor* (CP) connected to the switch via a *Switch Module Interface* (SMI). The CP configures the switch hardware to route incoming cells to the appropriate outgoing links by modifying tables within the switch, thus establishing connections. The switch routes signaling cells from clients or other nodes (as distinguished by the header) to the CP via port 0.

Figure 3 shows a generalization of the concept of a node, where more than one ATM switch is under the control of a single CP. The CP is directly connected to one of the switches, giving it direct access to the tables within that switch. Each of the other switches within the node (to which the CP is not directly connected) are termed *satellite switches*. Each of these has a slave control module that is able to modify the tables of its directly connected switch. These control modules operate under the direction of the CP, where the CP communicates with the satellite switches by sending ATM cells over the interconnection fiber optic links.

2.2 The ATM Standard

The emerging ATM standard [3][14] specifies link level cell formats for two interfaces: 1) the User-Network Interface (UNI), for communication between the client and the network, and 2) the Network-Node Interface (NNI), for communication between network nodes. The ATM UNI cell format is shown in Figure 4. It consists of a 48 byte information (data) field and a five byte header. The header has six fields: a Global Flow Control (GFC, 4 bits), a Virtual Path Identifier (VPI, 8 bits), a Virtual Channel Identifier (VCI, 16 bits), a Payload Type (PT, 3 bits), a Cell Loss Priority (CLP, 1 bit) and a Header Error Check (HEC, 8 bits).

Use of the GFC has not yet been standardize, although the intent is to use this field for arbitration on shared media access links (such as DQDB). The VPI and VCI fields are used to route cells. The ATM standard provides for two types of connections: *Virtual Path* (VP) and *Virtual Channel* (VC). With a VP connection, clients set the VPI field of the ATM header when sending cells. The network then uses the VPI for routing, possibly remapping this field at every switching node within the network, until the cells reach their destinations. With VC connections, clients set the VCI and the VPI when sending cells and the network uses both the VCI and VPI for routing. For VP connections, the VCI is preserved by the network—whatever value the sending client places in this field is delivered to the destination client(s) and is available for use by the clients, for example, as a multiplexing field. With VC connections, the VPI is not necessarily preserved by the network and may have to be set to a particular value (such as zero). Therefore, VC con-

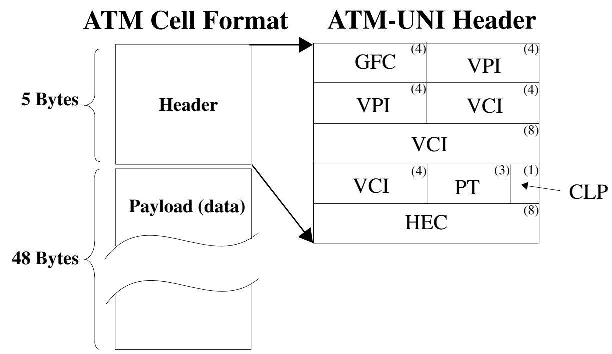


Figure 4. ATM UNI Cell Format.

nections do not allow the client to use the ATM header for multiplexing. The PT field is used to distinguish data cells from other cells, as shown in Table 2. The network congestion PT marking is used by the network to inform clients receiving a cell that congestion was encountered somewhere in the network. The tagged data PT marking can be used by the client to differentiate *special* cells. This marking is preserved by the network. One potential use is in delineating segmented frames, where the cell containing the last fragment of the frame is tagged and all other frames are untagged [9][42]. An example of this is shown in Figure 5 for IP datagrams. The CLP bit is used to mark low priority cells, where CLP=1 indicates low priority. This bit may be set either by clients or the network. The last header field, the HEC, is a cyclic redundancy check (CRC) on the header.

TABLE 2. Description of PTI Field Values.

PTI Value	PT Value (binary form)	Description
0	000	Client data, no congestion experienced by network
1	001	Tagged client data, no congestion experienced by network
2	010	Client data, congestion experienced somewhere in the network
3	011	Tagged client data, congestion experienced somewhere in the network
4	100	OA&M F5 link associated test cell
5	101	OA&M F5 end-to-end associated test cell
6	110	Resource management cell
7	111	Reserved for future use

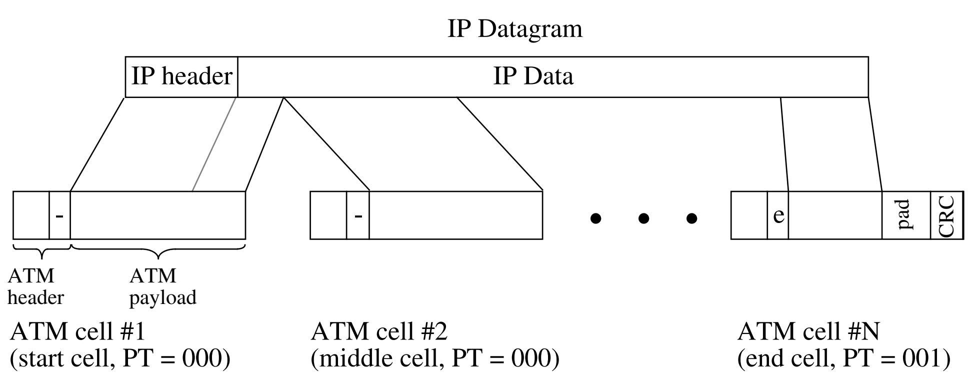


Figure 5. IP Segmentation/Reassembly Scheme.

2.3 ATM Cell Pipes

Clients of ATM networks communicate over what we term *cell pipes*. Our model supports *n*-way, bidirectional, multipoint cell-pipes, where point-to-point channels are simply a special case of multipoint channels. Figure 6 shows the conceptual view of a three endpoint cell pipe between clients A, B and C. All data sent by one client is received by all other clients who have elected to receive on this cell pipe. CMAP does not assume or require any particular ATM Adaptation Layer (AAL) on top of cell pipes. Rather, it provides clients with a *null-AAL* upon which any AAL can be layered (including none at all).

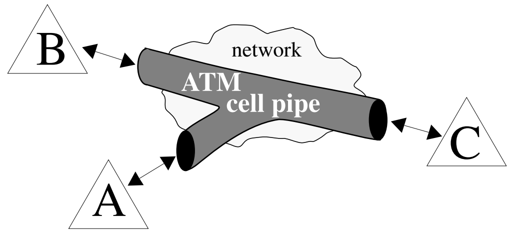


Figure 6. Three-way Cell Pipe.

2.4 Virtual Path and Virtual Channel Connections

In our call model, we allow clients to designate whether they want a virtual path or a virtual channel connection at connection setup time. Since, in VP connections, the VCI is delivered unchanged from the value assigned by the sender, the VCI can be used for source discrimination in multipoint connections, provided that each client places a unique VCI in all of its outgoing cells. Figure 7 shows an expanded view of the cell pipe from Figure 6, where the VCI is used for source discrimination. The figure shows three endpoints, A, B and C, communicating using a VP connection. Each endpoint is receiving and transmitting on the connection and all cells sent by a transmitter will arrive in the same order at each receiver. In the example, the clients have agreed that VCI 1 would be used for client A, 2 for B and 3 for C. Therefore, by using the VCI, all clients have the capability to distinguish the source of the transmitter.

With VC connections, the network uses the VCI to route the cells and may also translate the VPI on internal links (for intertrunk grouping, where multiple VC connections are carried on a preconfigured internal trunk). The VCI used by a transmitter, therefore, has no end-to-end significance. At some point during communication, all transmitters on a VC connection may have their cells interleaved. The cells arriving at a receiver will not be distinguishable by the ATM cell header—the VPI and VCI are the same for all arriving cells regardless of origin. Therefore, if VC connections are used for multipoint communication, higher level protocol information embedded in the ATM cell payload must be used for source discrimination (if required). VC connections are desirable for connections that do not need

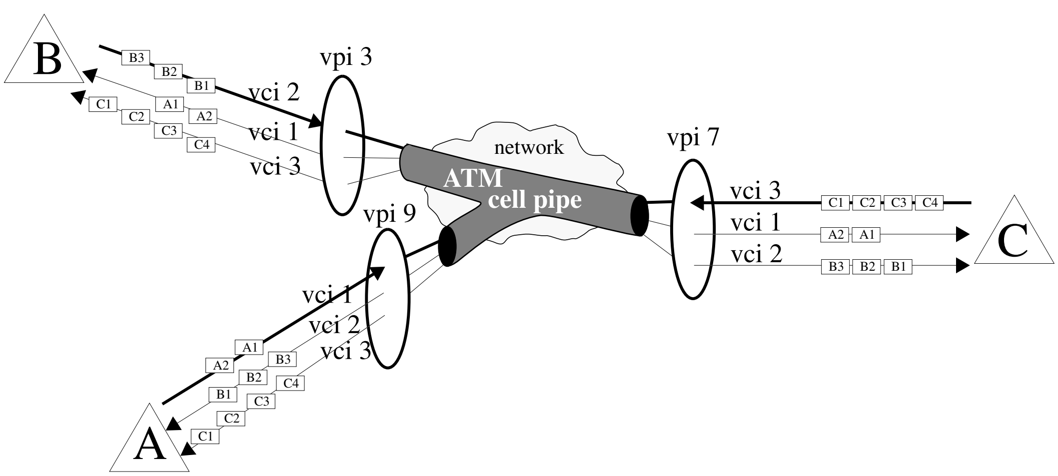


Figure 7. Sequenced Bidirectional Multipoint VP Connection Using the VCI for Source Discrimination.

source discrimination, and for connections that want to take advantage of rapid setup (where the network is able to reduce connection setup overhead by using preconfigured trunks).

2.5 VPI/VCI Assignment at the UNI

Our call model supports both *uni-directional* and *bi-directional* assignment of VPIs and VCIs at the UNI. Uni-directional assignment allows the VPI and VCI to differ for each direction of a *receive/transmit* connection. Bi-directional assignment requires the VPI/VCI values to be the same for both directions. In the case of uni-directional assignment, the receiving party on a link is responsible for assigning the VPI/VCI values that the sending party is to use. This policy was chosen since the tables that decode VPIs and VCIs in arriving cells are typically located at the receiver's site and managed by the receiver. When setting up a connection, a network entity (client or node) allocates a VPI/VCI table entry and informs the adjacent entity that he is to use the assigned VPI/VCI when sending cells to it. This resource management information is sent with the call/connection request. The adjacent network entity indicates the VPI/VCI for the opposite direction when it acknowledges the operation. Each entity allocates the resources according to this algorithm, obviating the need for VPI/VCI assignment collision detection, avoidance and recovery.

At the UNI, we allow clients to specify the VPI/VCI pair for both directions. In this way, a bi-directional assignment can be made by choosing identical values for each direction. The client can also leave both pairs empty, in which case the network will assign both (choosing identical values, if a strictly bi-directional assignment policy is chosen by the CMAP implementors).

3. Call Model

Network clients communicate with other clients by issuing CMAP requests that create and manipulate *calls*. A call is a distributed object maintained by the network that describes the communication paths that interconnect clients. The attributes of calls and the operations that can be performed on them define the *call model*. Initially, at call creation, a call has one or more communication channels, which we term *connections* (a connection is an end-to-end cell pipe, Section 2.3). The owner of a call (and other clients with the owner's permission) can add connections to or remove connections from the call. Multiple connections within a call are useful for applications such as video conferences, where one connection carries video and another audio. When a call is first created, the owner (and possibly one additional endpoint) are the only endpoints in the call. Additional endpoints may be added by: 1) invitation from the owner, where the invited party has the option of refusing the invitation, 2) request from a client not currently in the call to be added, where the owner has the option of denying the request, or 3) request from a third party, not necessarily in the call, to add a client, where both the owner and the client being added have the option to refuse. Calls are dynamic, in that the number of endpoints and number of connections in the call can vary over time, while communication is taking place. Additionally, the bandwidth reserved by a connection can also increase and decrease.

Our approach to the development of a call model for switched ATM networks is derived from a layered networking framework, as shown in Figure 8. We provide mechanisms for the use and maintenance of ATM network connections, but do not directly integrate higher level services into the model (such as NTSC to HDTV video conversion). Our model views all clients as *equal*. We draw no distinction, for example, between network gateways, multimedia workstations or video phones. All clients are constrained to access the network through our CMAP protocol and communicate through ATM cell pipes. We believe that our call model provides the necessary core functionality so that other services can be layered over CMAP, perhaps presenting a more sophisticated network model to users. Additionally, we feel that CMAP's interconnection services are suitable for both local and wide area networks.

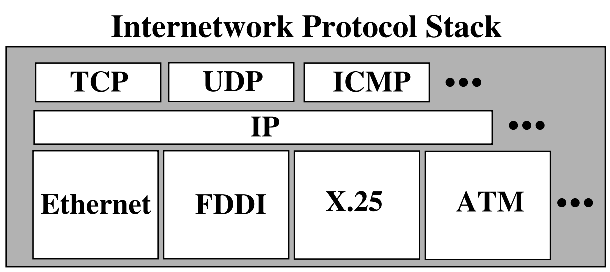


Figure 8. ATM as a Component Network in an Internetworking Environment.

3.1 Calls

The call is the primary object manipulated by clients. When a client creates a call or adds himself to an ongoing call, he becomes an *endpoint* in the call and gains access to all connections of the call. Figure 9 shows an example of a multimedia call at a client. Calls have a number of parameters that describe how clients may access the call, described in the following subsections and summarized in Table 3.

TABLE 3. Call Parameters

Parameter	Description
Type	Multipoint or point-to-point
Accessibility	Freedom (or lack thereof) with which clients can join the call
Monitoring	Level of notification of client joins and drops
Modifiability	Whether a nonowner client may add connections
Priority	Call level priorities, for call preemption
Connection List	Current connections in call

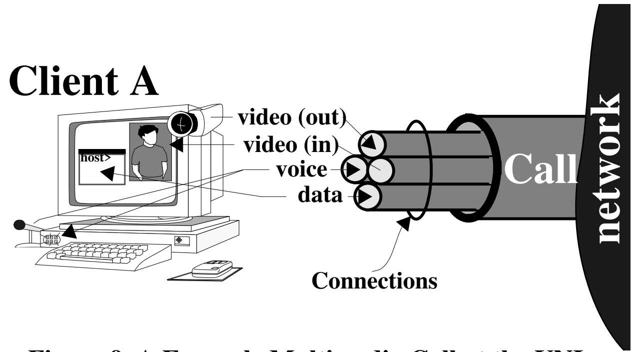


Figure 9. A Example Multimedia Call at the UNI.

3.1.1 Call Type

The call type designates whether the call is *multipoint* or restricted to *point-to-point*:

Type ∈ {**MULTIPOINT**, **POINT-TO-POINT**}

While the call model supports multicast calls as the general object, some calls are intrinsically point-to-point and, if designated as such, can be serviced more efficiently by the network (for example, by routing the call over internal trunks).

3.1.2 Call Accessibility

The accessibility parameter is a 1-tuple that describes how clients are allowed to join the call:

Accessibility ∈ {**OPEN**, **VERIFY**, **CLOSED**}

If the call is OPEN, any client in the network may join the call without the owner's permission. A CLOSED call restricts the call such that only the owner may add clients. If the call has the VERIFY accessibility, then any client may attempt to join the call, but the operation will be verified with the owner before it is allowed to complete.

3.1.3 Call Monitoring

The monitoring parameter is a 3-tuple designating whether endpoints of the call are notified when new endpoints join or drop out of the call:

Monitor = < **owner** ∈ {**ON**, **OFF**},
transmitters ∈ {**ON**, **OFF**},
all ∈ {**ON**, **OFF**} >

If the *owner* field is set to ON, all endpoint joins and drops are reported to the owner. Likewise, if the *transmitters* field is set to ON, all transmitters are notified when these changes occur, and if the *all* field is set to ON, all participants in the call are notified. Table 4 summarizes the meaningful combinations of call monitoring settings. When used in conjunction with an accessibility of OPEN, an owner modifiability setting of ON allows the owner to keep track of endpoint joins and drops without being burdened with explicitly verifying every such change.

TABLE 4. Summary of Monitoring Options.

Call Monitoring 3-tuple values	Monitoring state
< **OFF**, **OFF**, **OFF** >	No monitoring
< **ON**, **OFF**, **OFF** >	Owner monitoring only
< **OFF**, **ON**, **OFF** >	Transmitter monitoring only
< **ON**, **ON**, **OFF** >	Owner and Transmitter monitoring
< **any**, **any**, **ON** >	Monitoring for all endpoints

3.1.4 Call Modifiability

The modifiability parameter controls whether non-owners have the permission to add connections to the call:

$$\textbf{Modifiability} \in \{\textbf{ON}, \textbf{OFF}\}$$

Once a non-owner client has added a connection, ownership of the connection is passed to the call's owner. One example of where this option is useful is in multiparty conference calls, where each endpoint adds a one-to-all connection for his feed when joining the call.

3.1.5 Call Priority

The priority parameter is a 1-tuple that allows some calls to take precedence over others:

$$\textbf{Priority} \in \{\textbf{NORMAL}, \textbf{PREEMPT}, \textbf{OVERRIDE}\}$$

NORMAL is the lowest priority, followed by PREEMPT, followed by OVERRIDE. If the network does not have the resources to support a higher priority call and the required resources are currently allocated to lower priority calls, these resources are reclaimed (aborting the lower priority calls) so that the higher priority call can be supported. The intention is that most clients would only be allowed to specify normal priority calls, while selected clients (for example, the national weather service or the military) would be given the option of using one or both of the other priorities (as configured by network management).

The call level priority field affects how call control software processes the call and whether the call should be blocked if resources are unavailable. ATM cell level priorities are implemented separately as parameters of connections (Sections 3.2.1 and 3.2.2).

3.1.6 Call's Connection List

The connection list describes the connections of the call. The description includes allocated bandwidth, connection type, and receive/transmit permissions and mapping, as described in the following section. The list is dynamic since connections can be added to or removed from the call at any time, and since the attributes of connections can change over time.

3.2 Connections

The connection is the primary information carrying component of a call. The five parameters associated with all connections are described in the following subsections and summarized in Table 5. The first two of these parameters, type and bandwidth, are *global* connection parameters that apply to the entire connection and all endpoints in the connection. The third parameter, permission, has a global default value and *local* override values assigned to each individual endpoint. The fourth and fifth parameters, mapping and notification, are strictly local, where each endpoint sets these values independently from the other endpoints.

TABLE 5. Connection Parameters.

Parameter	Description
Type	Three tuple consisting of VP/VC, dynamic/static bandwidth and quality of service
Bandwidth	The reserved bandwidth for the connection (*best-effort* connections are supported)
Permission	Restrictions on the transmit/receive access of endpoints
Mapping	The current transmit/receive access mapping of each endpoint
Notification	Whether endpoints should be notified if the bandwidth changes

3.2.1 Connection Type

The connection type is a 3-tuple:

$$\begin{aligned}\textbf{ConnType} = \langle\ & \textbf{channel\_type} \in \{\textbf{VP}, \textbf{VC}\}, \\ & \textbf{BW\_type} \in \{\textbf{STATIC}, \textbf{DYNAMIC}\}, \\ & \textbf{QOS} \in \{\textbf{HIGH}, \textbf{MEDIUM} \text{ or } \textbf{LOW}\}\ \rangle\end{aligned}$$

The first option specifies the type of channel that the connection requires and must be one of two values: *virtual path* (VP) or *virtual channel* (VC), see Section 2.4. The second option specifies whether the connection is DYNAMIC or STATIC. If a connection is *static* then the bandwidth is fixed throughout the life of the call, which makes the connection more predictable and allows for more efficient use of network resources (such as routing over internal trunks). DYNAMIC connections can have their bandwidth modified during the life of the call. The last type option specifies the desired *quality-of-service* (QOS) and may have the values of HIGH, MEDIUM or LOW. QOS relates to options for cell loss behavior that may vary from network to network. QOS, therefore, is left intentionally vague. If the network can implement different cell loss behavior strategies then the network control software will group these into the categories of HIGH, MEDIUM and LOW.

3.2.2 Connection Bandwidth Specification

We support two types of bandwidth specification for connections: 1) *reserved*, where the network guarantees and enforces the requested bandwidth, and 2) *best-effort*, where the network neither guarantees nor enforces. Reserved bandwidth connections are treated preferentially by the network, whereas best-effort connections compete with one another for the remaining bandwidth not currently used by reserved connections.

The connection's bandwidth is specified as a 3-tuple, as shown below:

Bandwidth = < peak, average, peak_burst_length >

The *peak* and *average* parameters are expressed in cells per second. The *peak burst length* is measured in cells and indicates the maximum number of cells that the endpoint can send during a burst at peak rate. Best-effort connections are indicated by all zeros in the bandwidth specification.

For reserved bandwidth connections, the figures given are for aggregate bandwidth, as seen at any receiver in the connection (this takes into account the increased loading that results when the transmissions from multiple transmitters combine and flow over links traversed by the connection). Figure 10 illustrates how we use the bandwidth specification to characterize the endpoint's transmit behavior on a connection at the UNI. The *peak* value tells us a percentage of the bandwidth used by the endpoint when transmitting a burst of data. The user may not arbitrarily send cells back-to-back for the duration of the *peak burst length*, but must *pace* them according to the *peak* value as shown. The *peak burst length* tells the network how long the connection will remain transmitting at *peak* rates. The example in Figure 10 shows a *peak* value of 87,000 cells per second (in this case, roughly 25% of a 155Mbps link) and a *peak burst length* of 1000 cells. The *average* value is used over a longer period of time (for example, several seconds) to enforce the long term average utilization of the connection.

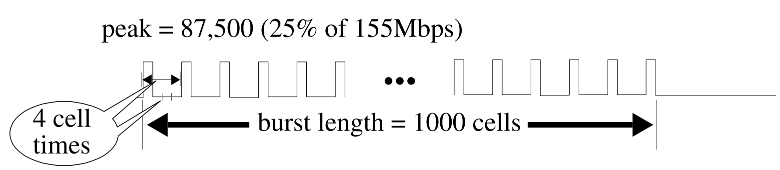


Figure 10. Cell Pacing and Peak and Burst Length at the UNI.

For best-effort connections, the endpoint is not restricted in any way. Instead, all cells are forcibly marked at the UNI by setting the CLP bit. Inside the network, as buffers fill, the network discards marked cells if there is no room in the buffers for unmarked (reserved) cells.

We chose to provide both reserved and best-effort bandwidth connections for several reasons. Some applications clearly need bandwidth guarantees, such as live video, where cell loss can result in incomprehensible transmissions. However, many other applications do not need these guarantees. For example, data transfer applications can usually tolerate lossy networks and network congestion using schemes such as retransmission and backoff, as in TCP/IP [18]. Additionally, many applications cannot easily predict their bandwidth requirements, and if forced to do so would produce specifications that are too low or too high, each with negative consequences. Finally, when all applications are forced to use reserved, guaranteed connections, network utilization is sometimes low (depending on the connection mix) since the level of allowed multiplexing must be limited to provide the guarantees. By introducing the best-effort bandwidth class, we accommodate those connections that do not need the guarantees and cannot easily estimate requirements, and we provide a means whereby the excess capacity of the network can be utilized.

3.2.3 Connection Permissions

Connection permissions describe the allowed receive and transmit capability of endpoints on the connection. Each connection has a *default permission*, which can be overridden by the call owner so that individual endpoints can be assigned different permissions. The permission tuple consists of a receive part and a transmit part, as shown below:

Perm = <Rcv = < perm ∈ {**ON**, **HOLD**,**OFF**},
changeability ∈ {**ON**, **VERIFY**, **OFF**}>,
<Trans = < perm ∈ {**ON**, **HOLD**,**OFF**}
changeability ∈ {**ON**, **VERIFY**, **OFF**}>>

For the receive part, a *perm* of ON indicates that the endpoint can receive on this connection, a *perm* of OFF indicates that the endpoint cannot receive on this connection, and a *perm* of HOLD indicates that the endpoint cannot receive, but that bandwidth should be reserved within the network up to the endpoint's access node (for use when the endpoint's receive permission is to be changed to ON at some later time). The *changeability* field indicates whether an endpoint can deviate from his assigned permissions when mapping the connection in, as described in the next section (Section 3.2.4). The transmit part is defined analogously to the receive part.

The default connection permission is a *global* connection parameter that applies to all endpoints in the connection, until overridden. Each endpoint also has an *endpoint permission* for each connection as a *local* connection parameter. The owner of the call can change individual endpoint permissions from that of the default value to give certain endpoints a restricted or expanded set of privileges. This can either be done when adding the endpoint to the call or at some later time by modifying the endpoint's attributes.

3.2.4 Connection Mappings

The connection mapping is a local connection parameter set by the endpoint and determines how the endpoint accesses the connection. The mapping tuple is defined as follows:

Mapping = < Receive = < perm ∈ {**ON**, **HOLD**,**OFF**}>,
< Transmit = < perm ∈ {**ON**, **HOLD**,**OFF**}>,
< Echo = < perm ∈ {**ON**, **OFF**}>>

Figure 11 shows six common ways (of the twelve possible) that an endpoint may map a connection to itself at its access point (refer to Table 10 on page 170 for a complete list of the possible mappings). The connection mapping is constrained by the connection permission. If the permission *changeability* field is set to OFF for the receive (transmit) part, the endpoint's receive (transmit) mapping must equal his permission *perm*. If *changeability* is ON, then the endpoint may choose any mapping. In this case, the permission is not enforced and is merely viewed as a suggestion to the endpoint. If the *changeability* is VERIFY, the endpoint can alter his mapping, but the owner will first be queried to see if the new mapping is acceptable. There is one exception to this general rule: if a receive or transmit permission *perm* is ON, the endpoint can freely change to HOLD regardless of the *changeability* setting.

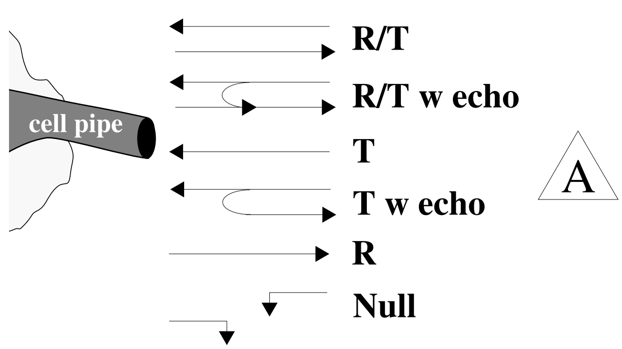


Figure 11. Sample Connection Mappings at the UNI.

The *echo* mapping is used in conjunction with transmit to allow endpoints to view their own transmissions. When set to ON, the endpoints's transmissions will be echoed and when set to OFF they will not be echoed. Endpoints who are allowed to transmit can freely change their echo mapping.

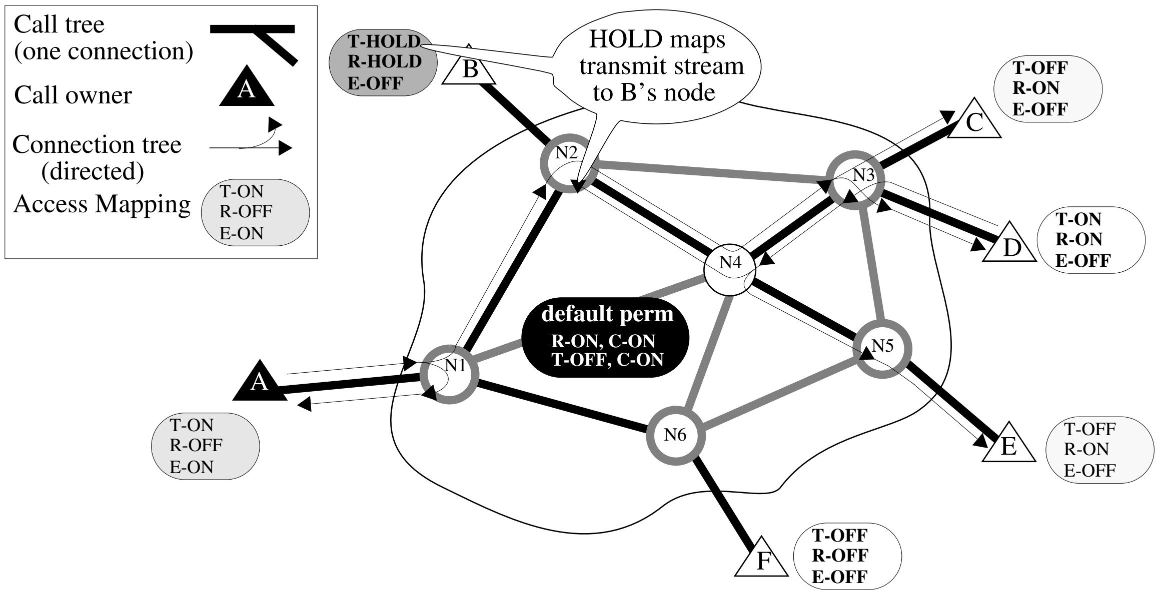


Figure 12. Example Call Tree and Client Access Mappings.

Figure 12 shows how a single connection call with a variety of endpoint access mappings would be implemented inside the network. The highlighted (dark black) links are those traversed by the call. The thin arrows show the links over which bandwidth is allocated (only in the directions of the arrows). As seen, bandwidth is not necessarily allocated along all links traversed by the call—it is only allocated along those links that flow from transmitters to receivers. Also, note how the HOLD receive and transmit mappings of endpoint B result in an internal flow to B as if B had mapped these to ON. If B subsequently changes his mappings to ON, the response time it takes to complete the request will presumably be much quicker than if the connections had not been present at B's access node and the blocking probability of the request will be lower since other connections cannot use this bandwidth.

3.2.5 Connection Notification

The notification flag is a local connection parameter set by each endpoint when joining a connection and has one of two values:

$$\textbf{Notification} \in \{\textbf{ON}, \textbf{OFF}\}$$

When notification is ON, the endpoint will be notified of all changes in bandwidth that occur on the connection. If OFF, the endpoint is not notified. This parameter is useful when a endpoint wishes to reconfigure an interface (for example, by adding more buffers) when a bandwidth change occurs.

4. Signaling the Network

Clients create, manipulate and delete calls by signaling the network using CMAP (Sections 6 and 7). CMAP messages are sent in-band over the ATM link connecting the client to the network. CMAP is layered over a transport service, referred to as CTL (CMAP Transport Layer), that provides reliable delivery of CMAP messages, and segmentation and reassembly (SAR) of CMAP messages into ATM cells (Figure 13). We define the general requirements for CTL, but do not stipulate a specific transport protocol or SAR protocol. As a result, CMAP can be layered over many existing protocols. The required and optional CTL functions are defined formally in Section 4.4 using the concepts introduced in Sections 4.1 through 4.3.

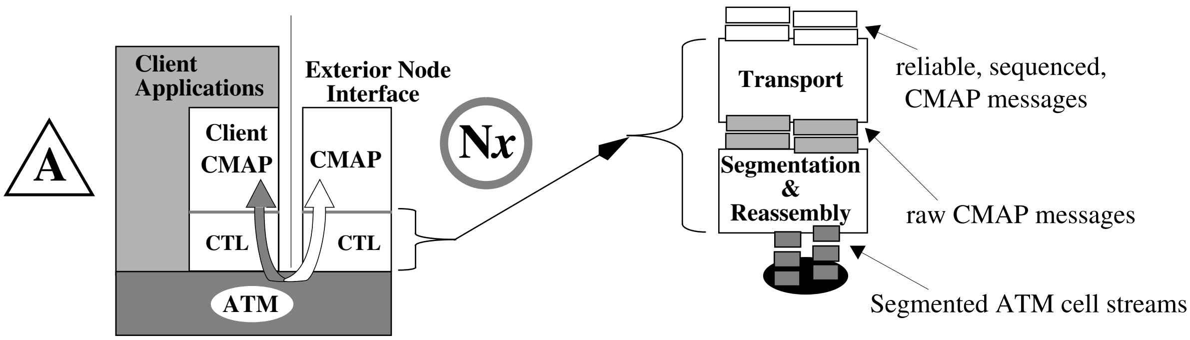


Figure 13. Client Access Protocol Stack for CMAP.

4.1 Signaling Connections

At the lowest level, the ATM cells comprising CMAP messages are sent over a well-known signaling connection, identified by a well-known VPI/VCI pair (Figure 14). One such bidirectional connection is setup between each client and the switch node's Control Processor (CP) when the node bootstraps. Since a node may consist of several interconnected switches all under the control of a single CP, this signaling connection may traverse multiple switches before terminating at the CP (as shown in Figure 14). Clients can request the creation of additional signaling connections, but are not required to do so (in this way our signaling connection functions as an ATM standard *meta-signaling connection*).

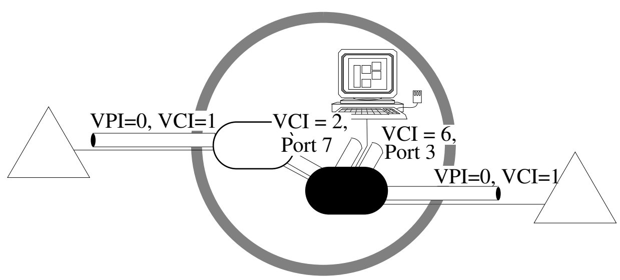


Figure 14. Signaling Connection at the UNI.

4.2 Multidrop Signaling

While we expect that there will typically be one client connected to each ATM switch access link in the ATM LAN environment, our model allows for multiple clients on the same link, as shown in Figure 15. This is one case where additional signaling connections may be desirable, so as to differentiate the multiple clients on the access link.

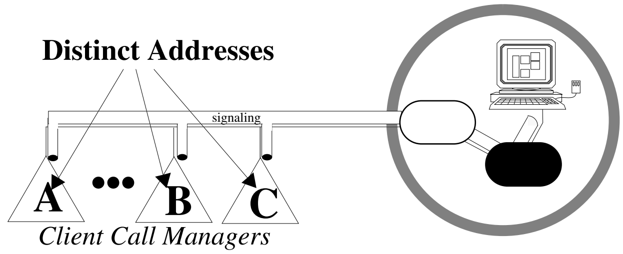


Figure 15. Multidrop Access at the UNI.

4.3 Surrogate Signaling

Another signaling capability that we allow for is *surrogate signaling*, where one client (called the *surrogate* client) is designated as the signaling entity for another client (called the *mute* client), as shown in Figure 16. The surrogate client originates all signaling messages for the mute client and receives all signaling messages from the network that would otherwise be sent to the mute client. Surrogate client signaling requires external configuration by network management. The *surrogate* client can be anywhere in the network and does not have to be local to the *mute* client's node.

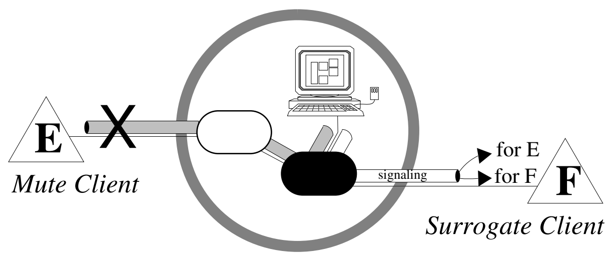


Figure 16. Surrogate Client Signaling at the UNI.

An example of where surrogate signaling could be used is shown in Figure 17, where an array of high resolution displays (used for digitized X-rays or other medical diagnostic displays) are controlled from a nearby control station. The physician uses the control station to select which images should be shown on each display and all of the signaling is performed by the control station.

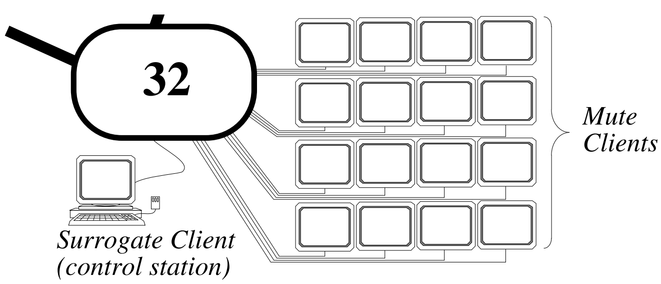


Figure 17. Surrogate Signaling for Medical Applications.

4.4 CTL Functionality

Table 6 lists both the required and optional CTL functions. The required functions are needed for correct CMAP operation. The optional functions can be used to augment client interfaces and applications. The first six of these functions are self explanatory, the remainder deserve further discussion. *Internal ping* provides CMAP with a mechanism to determine whether the remote peer is still alive. *Auto synchronization* does this automatically within the CTL layer, with notifications to the CMAP layer when synchronization is lost. *Multiple connections* on a single access link allow for multiple signaling connections to a single client and for multiple clients on an access link (multidrop support). *Query* allows for smart client applications who learn the capabilities of the network, then adapt to them. For example, an application could query to learn whether the network supports multipoint connections and, if not, it could emulate multipoint connectivity using several point-to-point connections, transparently to the user.

TABLE 6. Required and Optional CTL Functionality.

CTL Functionality	Description	Required/ Optional
SAR	Segmentation and Reassembly of CMAP frames to/from ATM cells	Required
Sequenced	Sequenced delivery of frames	Required
Lossless	No lost frames	Required
Duplicate suppression	No duplicate frames	Required
Error-free	Frames are delivered to higher level without bit errors	Required
Flow control	Throttling mechanism to handle network or host congestion	Optional
Internal ping	Internal CTL mechanism for learning if remote peer is still alive	Optional
Auto synchronization	CTL maintains synchronization between remote peers (using internal pings) and can report loss to higher layer (for example, to CMAP)	Optional
Multiple connections	Support for multiple CTL connections on a single access link	Optional
Query	Single ATM cell query to report SAR, transport protocol, etc.	Optional

5. Connection Management Network Protocol (CMNP)

Clients of BISDN networks create, manipulate and destroy calls by sending CMAP messages to exterior nodes of the network. These CMAP messages are transmitted in ATM cells, which are distinguished from data cells by sending them to the network via the signaling connection. The exterior nodes communicate with one another and with interior nodes by sending *Connection Management Network Protocol* (CMNP) messages. The CMNP messages are used to exchange, among other information, the VPIs and VCIs used to establish connections. Figure 18 shows a layered view of the protocol architecture.

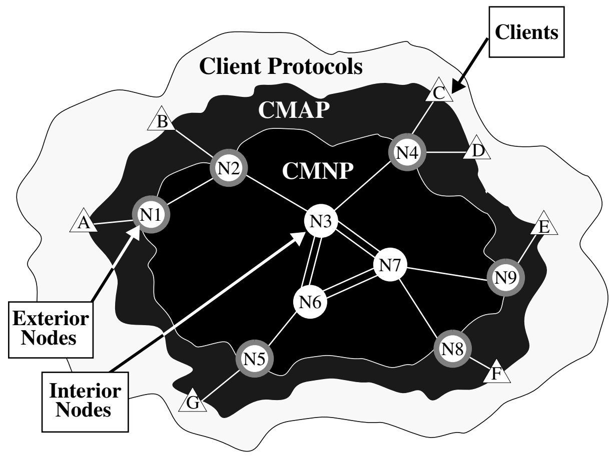


Figure 18. Layered View of the Protocol Architecture.

The client has no knowledge or visibility of CMNP. Since CMAP implementations can vary (Section 8), with different options supported in different implementations, the complexity and form of the CMNP protocol may vary between networks. For example, the CMNP protocol used to implement a point-to-point restricted CMAP network might be considerably different than the CMNP protocol used to support a CMAP network that supports *n*-way multipoint connections. CMNP is, therefore, not directly tied to CMAP and is specified separately.

6. Connection Management Access Protocol (CMAP) Overview and Concepts

In this section, we present an overview of our Connection Management Access Protocol (CMAP) and cover many of the underlying concepts required to fully understand the detailed CMAP operations (described in Section 7). CMAP is used by clients of the network to create, manipulate and destroy calls. All clients are constrained to access the network for connection management using CMAP and all clients, regardless of type, are considered equal.

This section is organized as follows. In Section 6.1, we briefly describe the CMAP operations and discuss the handshaking procedures used between clients and the network. In Section 6.2, we give three examples of how point-to-point and multipoint calls can be created using CMAP. In Section 6.3, we describe the addresses and identifiers used to label clients, calls, connections and endpoints within calls. Finally, in Section 6.4, we discuss how clients negotiate the attributes of calls and connections with the network.

6.1 Operations

A brief description of each CMAP operation appears in Table 7. The operations are divided into four classes: *commands*, *prompts*, *notifications* and *queries*. A *command* is an operation that a client uses to request a service from the network. The network responds to each command with a positive or negative acknowledgment, indicating whether the command succeeded. A *prompt* is an operation that the network uses to request an action by a client. Clients respond to prompts with an acknowledgment, then receive a confirmation from the network when the request is complete. A *query* is used by the network to request verification from a client on whether to proceed with an operation. The client responds to the query with an acknowledgment that indicates whether the operation should be allowed. Finally, a *notification* is used by the network to inform the client of a change in network or *call* state. No acknowledgment is required for notifications.

TABLE 7. CMAP Operations

Operation	Number of Phases	Description
Commands		
open_call	2	create a call.
close_call	2	terminate a call.
add_ep	2	add an endpoint to a call.
drop_ep	2	drop an endpoint from a call.
open_con	2	add a connection to a call.
close_con	2	drop a connection from a call.
mod_call	2	change call attributes.
mod_con	2	change connection attributes.
mod_ep	2	change an endpoint's access permissions or mapping.
change_owner	2	pass ownership responsibility to another endpoint in the call.
trace_call	2	ask the network to trace a call and report the call and connection attributes, and the endpoints participating in the call.
trace_ep	2	ask the network to trace an endpoint and report its attributes for a specific call.
client_reset	2	inform network that a client has reset (and lost all call state).

TABLE 7. CMAP Operations (Continued)

Operation	Number of Phases	Description
Prompts		
invite	2-3	invite an endpoint to join a call.
open_con_invite	2-3	invite an endpoint to join a new connection in a call.
mod_ep_invite	2-3	invite an endpoint to accept new endpoint permissions.
invite_owner	2-3	invite an endpoint to accept ownership responsibilities.
Query		
verify	2	ask call owner to allow addition of new endpoint or mapping change.
Notifications		
announce_ep	1	announce to endpoints the addition of a new endpoint.
drop_from_call	1	notify an endpoint that it has been dropped from a call.
announce_drop	1	notify owner that an endpoint has dropped out of the call.
announce_owner	1	notify an endpoint that a new owner has been established
announce_change	1	notify an endpoint of a change in call/connection attributes.
network_reset	1	inform clients that network node has reset (and lost all call state).

The "number of phases" column in Table 7 refers to the REQUEST, RESPONSE and CONFIRMATION messages exchanged as part of the operations, as alluded to above. All *commands* have two phases, REQUEST and RESPONSE, where the client sends a REQUEST to the network, then the network returns a positive or negative acknowledgment RESPONSE. Likewise, the *query* message has two phases, where the network initiates the REQUEST and the client returns a RESPONSE. *Prompts* have either two or three phases, depending on how the client responds. In both cases, the network initiates a REQUEST and the client returns a RESPONSE. If the client responds negatively to the network's request, the *prompt* operation terminates after the second phase. However, if the client responds positively, the network sends a CONFIRMATION to the client after the operation completes. *Notifications* have only a single REQUEST phase.

The following subsections briefly describe the CMAP operations. A more detailed description of each operation is contained in Section 7, along with message formats and state diagrams. Some of the operation descriptions below refer to *parameter negotiations*. These parameters are attributes of connections that can vary and that the client and network must agree on. Details of the parameters and the negotiation procedure are covered in Section 6.4.

6.1.1 Commands

6.1.1.1 open_call

The **open_call** operation is used by a client to create a call between himself and his access node. One or more connections can be created as part of the call using this operation, and another client can (optionally) be added as a second *endpoint* of the call. If another client is to be added, an **invite** operation is triggered, wherein the other client is asked to join the call. If so triggered, the network does not respond to the initiator of the **open_call** until the **invite** completes. The response indicates whether the call could be created by the network as specified (that is, whether the resources exist to support the connections and whether the additional endpoint accepted the invitation). The client creating the call is deemed the *owner* of the call, with special privileges, as described in the following subsections.

If the client issuing the **open_call** operation leaves some of the negotiable connection parameters undefined, an **invite** operation is triggered to the owner (before the second client is invited) so that values for these parameters can be negotiated.

6.1.1.2 drop_call

The **drop_call** operation destroys the call, removing all connections and reclaiming any resources used by the connections. Only the owner may issue the **drop_call** request. All other endpoints receive the **drop_from_call** notification to inform them that the call has been destroyed.

6.1.1.3 add_ep

The **add_ep** operation requests that an additional client be added to an ongoing call as a new endpoint. This operation can be initiated by the owner, by a non-owner endpoint in the call, by the client to be added, or by a third party client not in the call. In all cases except where the client issues the **add_ep** request for himself, an **invite** operation is triggered, wherein the new client is asked to join the call. As with **open_call**, the network does not respond to the initiator of the **add_ep** request until the new client has accepted or rejected the invitation. The response indicates whether the new client could be added, where a negative response is triggered if there are not enough network resources or if the new client rejects the **invite** request. If the **add_ep** succeeds, an **announce_ep** notification is sent to the owner, all transmitters, and/or all participants in the call, as per the call's monitoring setting (Section 3.1.3). The **add_ep** operation can be used to map a single client into a call multiple times, in which case each endpoint is treated as a logically separate client, each of whom may receive separate copies of the connections.

If a call's acessibility parameter is set to VERIFY (Section 3.1.2) and a client other than the owner issued the **add_ep** request, a **query** message is sent to the owner before the **add_ep** completes. Likewise, if the *changeability* part of the call's permission is set to VERIFY (Sections 3.2.3 and 3.2.4) and the endpoint changes its receive/transmit mapping from its receive/transmit permissions, a **query** message is sent to the owner. The owner responds with a positive or negative acknowledgment to permit or disallow the **add_ep** operation. If disallowed, the initiator of the **add_ep** request receives a negative response with an indication of why the request failed.

Normally, if a client issues an **add_ep** operation for itself, the **invite** operation is omitted. However, there is one exception to this rule: if the client leaves some of the negotiable connection parameters undefined, an **invite** is triggered so that these parameters can be negotiated.

6.1.1.4 drop_from_call

The **drop_from_call** operation removes a specified endpoint from the call. Either the owner or the endpoint to be dropped can issue this request. If the owner issues this request, the dropped endpoint receives a **drop_from_call** notification. If a non-owner endpoint drops out of the call, the owner, all transmitters, and/or all endpoints receive an **announce_drop** notification, as per the call's monitoring setting (Section 3.1.3). If the owner drops itself, the **drop_from_call** is treated like a **drop_call** operation, terminating the call.

6.1.1.5 open_con

The **open_con** operation requests that a new connection be added to an ongoing call. If the call's modifiability is ON (Section 3.1.4), any client can issue this request. Otherwise, only the owner can do so. All other endpoints in the call receive **open_con_invite** messages to invite them to join the new connection (unless the default receive and transmit permissions of the new connection are NULL). If any endpoint refuses the invitation or if the network resources required to support the new connection are not available, the **open_con** operation fails and a negative response is returned to the initiator of the **open_con**. Otherwise, a positive response is returned after the new connection has been established. Once established, ownership of the new connection transfers to the call owner.

Normally, the connection creator does not receive an **open_con_invite** request when initiating an **open_con** operation. However, if some of the negotiable connection parameters are left undefined, an **open_con_invite** to the creator is triggered before any of the other **open_con_invites** are sent out so that these parameters can be negotiated.

6.1.1.6 close_con

The **close_con** operation destroys a specified connection, reclaiming all resources used by the connection. Only the owner can issue this request. All other endpoints receive the **announce_change** notification to inform them that the connection has been destroyed. If this is that last connection in a call, the **close_con** request is treated like a **drop_call**.

6.1.1.7 mod_call

The **mod_call** operation requests modification of the attributes of the specified call (Section 3.1). Only the owner can perform this operation. The response indicates whether the operation succeeded.

6.1.1.8 mod_con

The **mod_con** operation requests modification of the attributes of the specified connection (Section 3.2). Only the owner can perform this operation. The response indicates whether the operation succeeded.

6.1.1.9 mod_ep

The **mod_ep** operation is used to modify an endpoint's permissions, mapping and/or notification flag for specified connections of the call. Only the owner may modify the permissions, whereas both the owner and the endpoint can modify the mapping and notification flag. If the owner generates the **mod_ep** request and changes the endpoint mapping or notification flag, a **mod_ep_invite** is sent to the affected endpoint. If the endpoint negatively acknowledges the **mod_ep_invite** or changes the mapping to an illegal value, the endpoint is dropped from the call and the **mod_ep** request fails with an indication of the reason for failure returned to the initiator. If the **mod_ep** succeeds and the endpoint's receive mapping changes on any connection, an **announce_ep** notification is sent to the owner, all transmitters, and/or all participants in the call, as per the call's monitoring setting (Section 3.1.3).

Normally, if a client issues an **mod_ep** operation for itself, the **mod_ep_invite** operation is omitted. However, there is one exception to this rule: if the client leaves some of the negotiable connection parameters undefined, the **mod_ep_invite** is triggered so that these parameters can be negotiated.

6.1.1.10 change_owner

The **change_owner** operation is used to transfer ownership of the call to another endpoint. Only the owner can issue this request and the other endpoint must already be in the call. An **invite_owner** operation is triggered, where the designated new owner has the option of refusing the change. If accepted, the **change_owner** is positively acknowledged to the old owner and the **announce_owner** notification is sent to all participants in the call to inform them of the change.

6.1.1.11 trace_call

The **trace_call** operation is used to obtain the attributes of the call and of the connections that comprise the call, and a list of the endpoints participating in the call. Currently, any endpoint can issue this request, although this ability may be restricted in the future (perhaps dependent on the call accessibility, Section 3.1.2).

6.1.1.12 trace_ep

The **trace_ep** operation is used to obtain detailed information about a particular endpoint participating in a call, including the endpoint's permission and mapping for all connections of the call. As with **trace_call**, any endpoint can issue this request, although this ability may be restricted in the future.

6.1.1.13 client_reset

The **client_reset** operation is used by a client to inform the network that it has reset. The network performs a **drop_call** on all calls that were owned by the client and a **drop_from_call** on all calls that the client was participating in as a non-owner.

6.1.2 Prompts

6.1.2.1 invite

The **invite** prompt is triggered by the **open_call** and **add_ep** operations to invite a new endpoint to join an ongoing call. The attributes of the call (including the number of connections, connection bandwidths, connection mappings and the VCIs and VPIs to be used by the endpoint for the connections) are presented to the new endpoint. If the new endpoint responds negatively, the associated **open_call** or **add_ep** that triggered the **invite** fails. If the new endpoint responds positively, the network will attempt to connect the endpoint to the call (resources permitting). If the connection succeeds, a positive confirmation is sent to the new endpoint; otherwise, a negative confirmation is sent.

6.1.2.2 open_con_invite

The **open_con_invite** prompt is triggered by the **open_con** operation and is sent to all endpoints in the call with non-NULL receive/transmit permissions on the new connection. As with **invite**, the endpoint is given the connection parameters and an opportunity to accept or reject the invitation. If the invitation is rejected, the **open_con** operation fails.

6.1.2.3 mod_ep_invite

The **mod_ep_invite** prompt is triggered by the **mod_ep** operation when the endpoint's receive/transmit mapping is changed. If the endpoint responds negatively or changes the mapping to an illegal value, the endpoint is dropped from the call and the **mod_ep** operation fails with the reason reported to the initiator.

6.1.2.4 invite_owner

The **invite_owner** prompt is triggered by a **change_owner** operation. The designated new owner is invited to take over the ownership privileges and responsibilities of the call. A negative response causes the **change_owner** operation to fail.

6.1.3 Query

6.1.3.1 verify

The **verify** query is sent to the owner of a call to verify an **add_ep** request. A **verify** query is triggered under two circumstances: 1) when the call's accessibility is set to VERIFY (Section 3.1.2) and the request was issued by a client other than the owner, and 2) when the *changeability* part of the call's permission is set to VERIFY (Sections 3.2.3 and 3.2.4) and the endpoint changes its receive/transmit mapping from its receive/transmit permissions. If the owner responds positively, the **add_ep** operation continues. Otherwise, the **add_ep** operation aborts, with a negative response sent to the initiator of the **add_ep** opeartion and a negative confirmation sent to the invited endpoint (if an **invite** was triggered).

6.1.4 Notifications

6.1.4.1 announce_ep

The **announce_ep** notification is sent to the call's owner, all transmitters in the call, and/or all endpoints in the call (as per the call's monitoring setting, Section 3.1.3) when a new endpoint joins the call (via **add_ep**) or changes its receiver mapping on any connection in the call (via **mod_ep**).

6.1.4.2 drop_from_call

The **drop_from_call** notification is sent to an endpoint when it is removed from a call by another participant as a result of a **drop_call** or **drop_ep** operation.

6.1.4.3 announce_drop

The **announce_drop** notification is sent to the call's owner, all transmitters, and/or all endpoints (as per the call's monitoring setting, Section 3.1.3) to inform those notified that an endpoint has issued a **drop_ep** request, thereby leaving the call.

6.1.4.4 announce_owner

The **announce_owner** notification is sent to all endpoints in a call after a **change_owner** operation has successfully completed to inform the endpoints of the identity of the new owner.

6.1.4.5 announce_change

The **announce_change** notification is sent to an endpoint to inform it of: 1) a **drop_con** operation, dropping one of the connections in the call, 2) a **mod_con** operation, changing the bandwidth or other attributes of the connection, or 3) a **mod_call** operation, changing attributes of the call.

6.1.4.6 network_reset

The **network_reset** notification is sent to a client when a network node losses all state information about the calls the client was participating in (for example, due to a hard reset in the network). The client is automatically dropped from all of these calls.

6.2 Examples

This section describes, through three examples, how common CMAP operations interact by demonstrating how a point-to-point, a many-to-many and a one-to-many call could be established. Note that the transition from point-to-point to multipoint is *seamless* in that the operation that adds the third endpoint (**add_ep**) is the same operation that added the second endpoint.

6.2.1 Point-to-point Call

Figure 19 shows the operations performed when a point-to-point call containing two connections is established between clients A and B. In step 1, an **open_call** request is issued by A. Included in the request is all of the call and connection attributes, including the bandwidth requirement for the connection, the default receive/transmit permissions of the endpoint(s) to be added, and the call's accessibility and monitoring characteristics (Section 3). Upon receiving the request, node N1 makes sure the required resources are available, then responds to A with an acknowledgment (step 2), creating a call between A and N1. In steps 3 and 4, a second connection is added to the call when A issues an **open_con** request and receives an acknowledgment. In step 5, A sends an **add_ep** request to the network. In step 6, the network sends an **invite** prompt to B, asking B if it would like to join the call. This request contains the bandwidth and receive/transmit characteristics of the call so that B can decide whether it can accept the call. Assuming B has the required resources and the desire to communicate with A, B acknowledges the **invite** in step 7. Finally, the network confirms that B has been added in step 8 and acknowledges A's **add_ep** request in step 9. These last two messages occur concurrently and may be received in reverse order.

Neither party should transmit until receiving its last response from the network since the connection is not known to be established until this time. Since messages in steps 8 and 9 can be reordered, either party could begin receiving data before receiving its last response. Therefore, client A must be prepared to receive data from B as soon as the **add_ep** message is sent (step 5) and client B must be prepared to receive data as soon as the **invite** acknowledgment is sent (step 7).

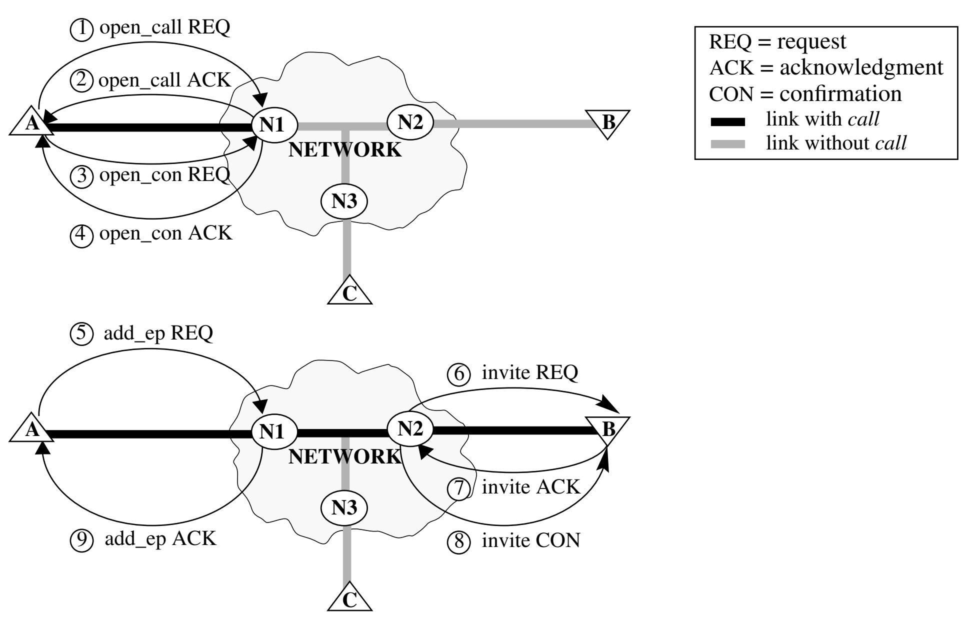


Figure 19. Building a Point-to-Point Call.

The **open_call** operation allows the client to create as many connections as he wishes int the **open_call** request. The client may also request one additional endpoint in the **open_call**. It follows, therefore, that the addition of the second connection to the call in steps 3 and 4, and the invitation extended to client B in step 5 to join the call, could have been accomplished in one **open_call** operation as outlined in Figure 20 below. The **invite** to B must still occur, but it is initiated by the **open_call** instead of the **add_ep** as in Figure 19.

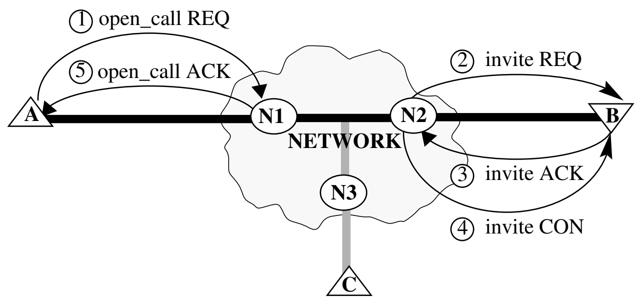


Figure 20. Building a Point-to-Point Call With One open_call Operation.

6.2.2 Many-to-Many Call

The previous example is extended to a many-to-many call by issuing an **add_ep** request for client C. All three clients, A, B and C**,** can issue the request, although if the accessibility is *closed*, requests by B and C will be denied, and if the accessibility is *verify*, A will be asked if the addition is acceptable before C is added. Figure 21 shows the steps when C attempts to add itself, assuming *verify* accessibility and monitoring by the call owner, client A. In step 1, C sends the initial **add_ep** request. In steps 2 and 3, A receives the **verify** request from the network and acknowledges the request (granting C permission to join). In steps 4 and 5, the network adds C and responds to C with an **add_ep** acknowledgment, then sends an **announce_ep** notification to A. As in the point-to-point case, client C must be prepared to receive data immediately after issuing the **add_ep** request.*

Additional endpoints may be added while existing endpoints are communicating. In this case, the new endpoints can start receiving data as soon as the connection is established at their access termination, perhaps before receiving the **add_ep** acknowledgments.

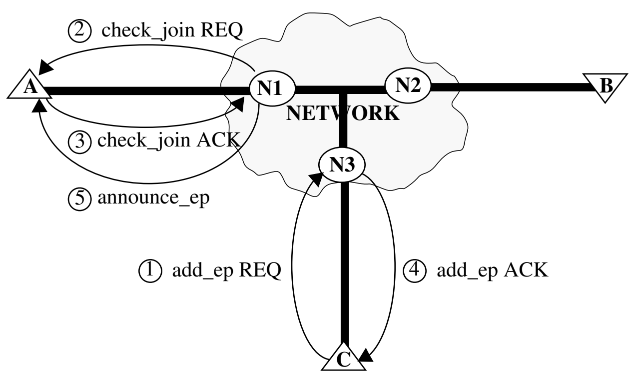


Figure 21. Building a Many-to-Many *Call*.

6.2.3 One-to-Many Call

A one-to-many call is created and manipulated in the same manner as a many-to-many call, where the permissions at call creation are set so that the owner or another designated endpoint in the call is the only endpoint that transmits and all other endpoints can only receive. The main differences in the way these calls are handled occur within the

* There are exceptions to this rule, a client adding itself may elect to be **invite**'d to a call by omitting certain negotiated fields, as discussed in Sections 6.1.1.3 and 6.4.

network, where bandwidth is only allocated in the direction of the transmitter to the other endpoints, rather than in both directions, as with a many-to-many call.

If a one-to-many call is used to support broadcast video, the network clients would presumably issue **add_ep** requests when they wish to join the call (as in the previous example, Section 6.2.2). If the broadcast source is *pay-for-view*, the owner might create the call with *verify* accessibility. Otherwise, perhaps for a free broadcast channel, the call might be created with the *open* accessibility.

6.3 Addresses and Identifiers

Clients, calls, connections and the endpoints participating in calls all require some form of identification at the CMAP level so that these entities can be referenced by clients. *Client addresses* are unique labels assigned to clients so that clients can identify one another. Client addresses are also used by the network to locate clients and route messages internally. *Call identifiers* are unique labels assigned to calls so that each call can be referenced by clients (for example, for joining and modifying calls). Clients can assigned well-known call identifiers to calls that are to be globally known (for example, for broadcast video distributions). *Connection identifiers* are unique labels for the connections within a call so that clients can individually reference the connections (for example, for changing ones receive/transmit mapping on a given connection). *Endpoint identifiers* are labels assigned to each client participating in a call. A client is allowed to join a call more than once and the endpoint identifier is used to distinguish the separate instances of client participation (for example, so that the client can drop one instance of the call while retaining one or more others). Detailed descriptions of all of these identifiers appear in the following subsections.

6.3.1 Client Addresses

Each client of the network is identified by a unique address. This is true as well for multiple clients who share a common multidrop access link (Section 4.2). These addresses are used in CMAP operations to indicate the client to be added, modified or dropped from a call or connection, and to indicate the client performing the operation. Internally, the network uses the addresses for routing.

The CMAP protocol does not specify a particular addressing scheme. Rather, CMAP implementations can choose to support one or more addressing schemes (for example, an implementation might support well known addressing schemes, such as IP or NSAP, and one or two experimental addressing schemes).

The format of a client address is shown in Figure 22. The address size is a constant 24 bytes, regardless of the addressing scheme used. The first four bytes comprise a *type* field that tells how the remainder of the address should be interpreted. The remainder is partitioned into three additional fields whose sizes vary depending on the type of addressing used: a *network address field*, *local address field* and an *unused* field. Three example partitionings are shown in Figure 23. The network address field contains the only portion of the address that the network uses in client identification and routing. The local address field is passed end-to-end in CMAP operations for use by the clients and is not interpreted by the network. The remainder of the address is unused.

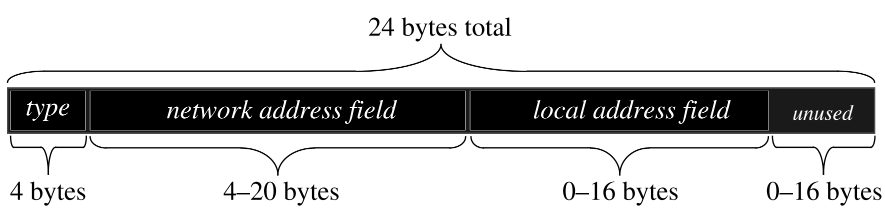


Figure 22. Client Address Format

The first diagram of Figure 23 shows the partitioning used for IP addresses [18]. The network address field is 4 bytes and holds the IP address. The local address field is 2 bytes and contains the higher level port number (for example, the TCP or UDP port). The remaining 14 bytes of the address field are unused. CMAP networks using IP addresses use only the network address field for routing. The local address field information may by used by clients to identify a particular service (port) at a client to which an operation is directed. The second diagram of Figure 23 shows the partitioning for the CCITT E.164 addresses [13] used in the public telecommunications networks. The network address field is 8 bytes and encodes the 15 decimal digit E.164 address. The local address field is 4 bytes and holds the 4 byte E.164 subaddress field. The remaining 8 bytes are unused. The bottom diagram of Figure 23 shows the partitioning for OSI NSAP (Network Service Access Point) addresses [17]. The network address field is 19 bytes and contains all fields of the NSAP address except for the SEL (selector) field, which is contained in the 1 byte local address field. Since NSAP addresses are 20 bytes in total length, there is no unused portion.

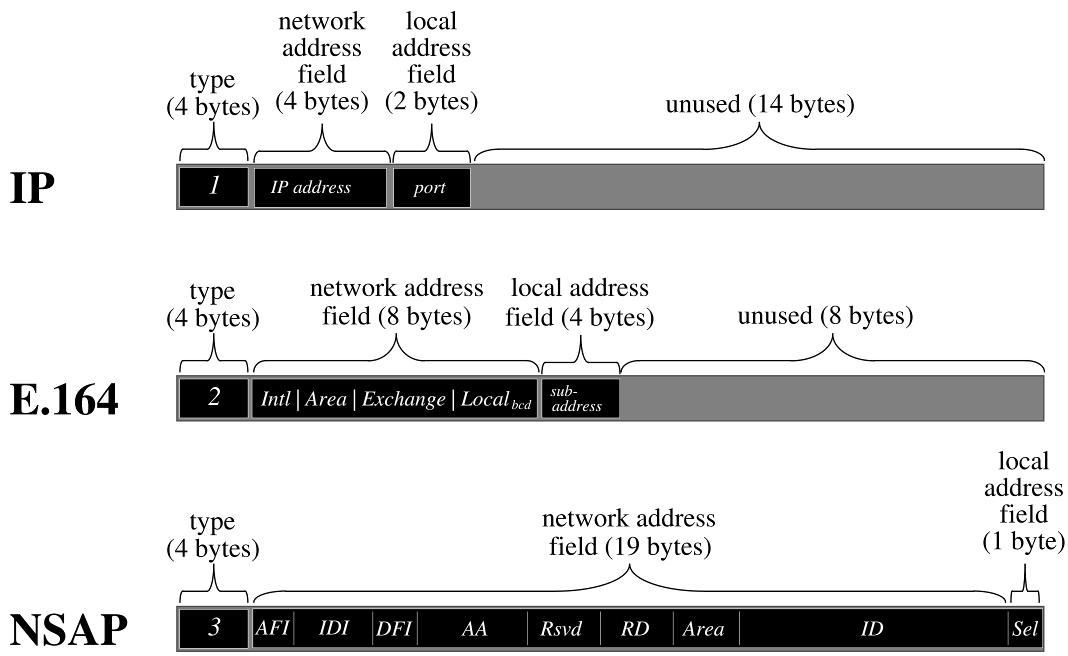


Figure 23. Examples of Client Address Partitioning.

6.3.2 Call Identifiers

Every call in the network is assigned a unique call identifier (call_id). The call identifier is the concatenation of the address of the client who created the call and a *local call identifier*, as shown in Figure 24. The local call identifier (lcid) is specified by the creator of the call at the time of call creation (in the **open_call** operation). The network checks the lcid to make sure that this client is not using it for another active call. If not, the call is assigned the lcid and the request proceeds. Otherwise, the request is negatively acknowledged. Call identifiers are guaranteed to be unique since client addresses are unique and since a given client can only use a given lcid for one active call. Clients can create well-known call identifiers for particular services (such as a broadcast video distribution feed) by assigning specific lcids to these services. Other clients in the network then join into these well-known services by using the well-known client address and lcid combination.

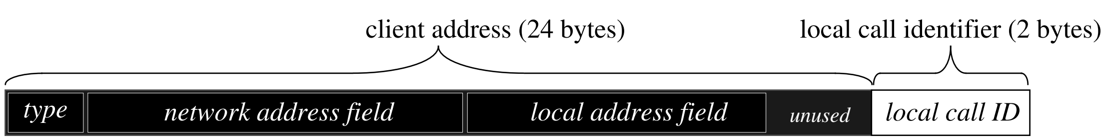


Figure 24. Call Identifier (call_id).

6.3.3 Connection Identifiers

The connection identifier (con_id) is a two byte field that uniquely labels each connection within a given call. Connection identifiers are used by clients in CMAP requests to specify the connection or connections of the call to be operated upon (for example, modified or deleted). Clients that create connections can (optionally) specify the connection identifier to be assigned to the connection. The network guarantees that no other connection has the same identifier and, if one does, the network negatively acknowledges the client's creation request. Alternatively, clients can allow the network to choose the connection identifier, wherein the network always chooses a unique one. As with well-known call identifiers, clients can create well-known connection identifiers for particular services within a call.

6.3.4 Endpoint Identifiers

Clients of the network are allowed to join calls more than once, in which case they have multiple *endpoint mappings* of the same call. The network treats each endpoint mapping as if it were associated with a separate client. That is, each mapping specifies receive/transmit access for the connections of the call and the network sends data streams to each endpoint as per the mapping. As a result, a client who has mapped a connection in more than once can receive multiple copies of the data streams associated with the connection.

Each endpoint mapping for a given client is assigned a unique two byte endpoint identifier (ep_id). Endpoint identifiers are only unique for a given client since they are only used to distinguish multiple client endpoint mappings. A client can direct a CMAP request to a particular client endpoint by specifying the client's address and the endpoint identifier pair corresponding to that endpoint mapping.

6.3.5 Identifier Example

Figure 25 shows an example of two network clients on a shared access link (with addresses A and B) participating in two calls. Client A owns one call ( ) with the assigned call identifier A:3. Client A is mapped into this call once with the endpoint identifier 1. Client B is also mapped into A's call and also has the endpoint identifier 1 (endpoint identifiers only need to be unique for a given client). Client F (not shown) owns the second call ( ) with the assigned call identifier F:1. Client B is mapped into F's call twice, once with the endpoint identifier 1 and once with the endpoint identifier 2. Internally, client B has mapped call A:3 to one video phone and call F:3 to both video phones.

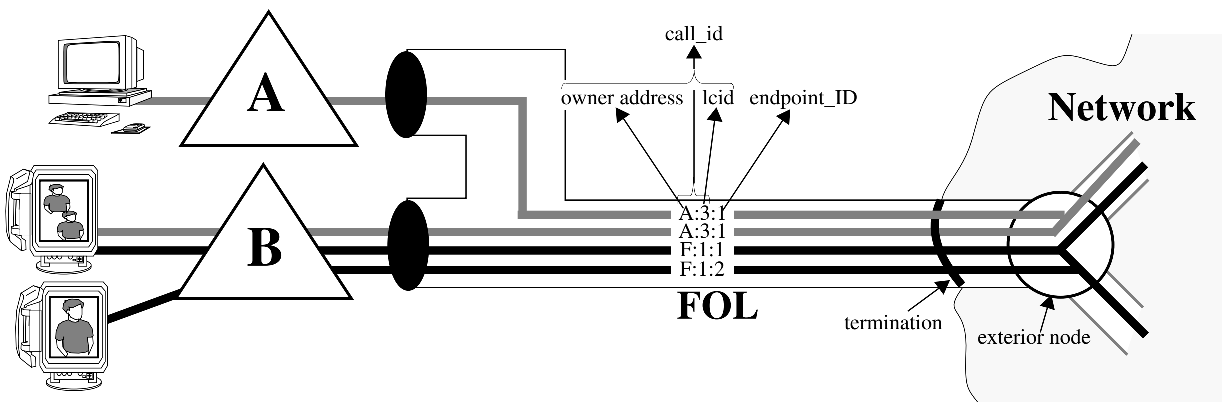


Figure 25. Differentiating Calls at the UNI

6.4 Parameter Negotiation

Many of the call and connection attributes (including the connection identifier described in Section 6.3) must be agreed upon by clients and the network. These parameters are negotiable in that either the client can choose the values or let the network choose. For the CMAP *commands* that bring calls, connections and endpoints into existence, the client can either specify initial values for the negotiable parameters in the *request* phase of the operation, or leave the values blank and allow the network to choose. If the client chooses values, the network checks them to make sure that they do not conflict, negatively acknowledging them in the *response* phase if they do. If the client leaves the values blank, the network chooses values and returns them to the client. For CMAP *prompts* (such as an **invite**) that add an endpoint to a call or that map a connection into an endpoint, the network always chooses values for the negotiable parameters, but the client can override the network's choices. If the client overrides, the network checks the new values and negatively acknowledges the response if illegal or conflicting values were chosen.

6.4.1 Negotiable Parameters and The Negotiation Procedures

Table 8 lists the negotiable parameters and the operations in which they are negotiated. The connection identifier (con_id) parameter was described in Section 6.3. The endpoint mapping (ep_map) and creator mapping (crtr_map) parameters refer to the receive/transmit mapping of the endpoint being added to the connection and that of the creator, respectively (Section 3.2). The VPI and VCI parameters (Section 2) are defined for each connection, where a transmit VPI/VCI pair identifies the VPI and VCI that the endpoint will use when transmitting data on this connection to the network and the receive VPI/VCI pair identifies the VPI and VCI that the network will use when sending data on this connection to the endpoint. For VP connections, the VCI is unneeded and its specification is, therefore, ignored in CMAP operations. Likewise, for connections mapped in as NULL for either the receive or transmit direction, the VPI/VCI pair for this direction is ignored.

TABLE 8. Parameter Negotiations:

C – Client may specify, assigned by network if unspecified

N – Network initiated negotiation

B – Client may specify, network initiated negotiation if unspecified

M – Client must specify, but can be overridden during network negotiation

operations\fields	con_id	ep_map	ctrt_map	VPI/VCI
open_call	C		M	B
open_con	C		M	B
add_ep		M (self) N (other)		B (self) N (other)
mod_ep		M (self) N (other)		B (self) N (other)
invite		N		N
open_con_invite		N		N
mod_ep_invite		N		N

In Table 8, a table entry of **C** indicates that the client either chooses a value for the given parameter and specifies it in the *request* phase of the given CMAP command, or the network assigns a value and reports it to the client in the *response* phase, where the client must accept the network's choice. If the client makes an initial choice and the value is illegal, the operation fails (signalled by a negative response from the network). A table entry of **N** indicates that the network initiates the negotiation for the given parameter by specifying a value in the *request* phase of the given CMAP prompt and the client can accept or override the network's choice in the *response* phase of the prompt. If the client overrides the network's choice, the negotiation is completed in the *confirmation* phase of the prompt, where the network either accepts or rejects the client's choice. A table entry of **B** indicates that the client either chooses a value for the given parameter and specifies it in the *request* phase of the given CMAP command, or the client leaves the value blank and a network negotiation is triggered, wherein the network sends an **invite** prompt back to the client with an initial value. As with **C** parameters, if the client makes an initial choice and the value is illegal, the operation fails. If the client leaves the parameters blank, a network initiated negotiation proceeds as with **N** parameters. A table entry of **M** indicates that the client must specify a value for this parameter in the initial request, but if a prompt is triggered to negotiate a different missing parameter (a **B** parameter), the **M** parameter can also be changed. The **add_ep** and

mod_ep negotiations function differently depending on whether a client issues one of these operations to add or modify himself, or to add or modify another client in the network. When adding or modifying another client, the negotiation is carried out between the client to be added or modified and the network (not between the client initiating the request and the network). Operations in Table 8 with no letter do not have a negotiation for the corresponding parameter.

6.4.2 Negotiation Steps for Each Operation

For the **open_call** operation, if the con_id is left blank, the network assigns a value and returns it in the response. If the **open_call** operation contains multiple connection specifications and several con_ids are blank, the connection specifications are returned in the response in the same order as they were specified in the request (with assigned con_ids) so that the client can differentiate the connections. If the VPI/VCI pair is left blank, an **invite** operation is triggered and sent to the call creator for the purpose of negotiating these values (see the description of the **invite** negotiation below for details). In any case, if another endpoint is specified as a second participant in the call, this client is invited and a second negotiation is carried out between the network and this client to negotiate the parameters applicable to the second client's access in the call.

The **open_con** negotiation operates analogously to the **open_call**, with the difference being that an **open_con_invite** (rather than an **invite**) is triggered to the client performing the **open_con** if the VPI/VCI pairs are left blank. An **open_con_invite** is also sent to all other endpoints in the call for the purpose of negotiating their access parameters (ep_map and VPI/VCI pairs) for this connection (unless the default receive and transmit permissions for all other endpoints in the call is NULL, in which case these negotiations are not required).

For the **add_ep** operation, the client can either invite himself or another client to join the call. If he is adding himself and the VPI/VCI pairs are left blank, an **invite** operation is triggered to negotiate these values. If the client is adding another client, the other client is sent an **invite** to negotiate the parameters for use between the other client and the network. On the **add_ep**, the client has the option of omitting all connection specifications (in effect saying that he does not know what connections are in the call). In this case, an **invite** is triggered with defaults for all parameters filled in, wherein the client can negotiate the values.

The **mod_ep** negotiation operates analogously to the **add_ep** operation, with the difference being that a **mod_ep_invite** (rather than an **invite**) is triggered to negotiate the parameters.

When the network generates an **invite** operation, initial values for all negotiable parameters (ep_map and the VPI/VCI pairs) are chosen by the network and sent in the request phase of the **invite**. Initial values for these parameters can be suggested in the **open_call** or **add_ep** request that triggered the **invite**, in which case the network uses these values, provided that they are legal. If the client wishes to accept the parameters sent from the network in the **invite** request, he echoes them back to the network in the **invite** response, thus terminating the negotiation. The network later responds with an **invite** confirmation, indicating whether the operation succeeded. If the client does not like the network's suggestions, he can change any or all of these parameters and send the new parameters back in the response. If the client specifies illegal values for any of the parameters, the network sends a negative confirmation to the client and the operation that triggered the **invite** fails. Otherwise, a positive confirmation is returned (assuming the remaining internal steps of the operation succeed) and a response is sent to the client that initiated the **open_call** or **add_ep** operation that triggered the **invite** with the renegotiated values actually used by the added client.

The **open_con_invite** and **mod_ep_invite** negotiations proceed analogously to the **invite** negotiations. Initial values for the negotiable parameters can be suggested in the **open_con** and **mod_ep** requests that trigger these operations.

Note that when an **invite** (**open_con_invite**) is triggered by an **open_call** (**open_con**), the ep_map in the **invite** (**open_con_invite**) request corresponds to the unspecified ctrt_map parameter in the **open_call** (**open_con**) request.

More details on parameter negotiation, as well as the definition of *blank* values for the different parameters, are contained in Section 7 and Appendix B.

6.4.3 Examples

This section contains examples of parameter negotiations for selected CMAP operations. Three examples are given: 1) negotiation of connection identifiers (con_ids) in the **open_call** operation, 2) negotiation of endpoint mapping (ep_map) in the **add_ep** operation, and 3) negotiation of ep_map in the **open_con** operation.

6.4.3.1 con_id negotiation in the open_call operation

Figure 26 shows the negotiation of con_ids in the **open_call** operation. In step 1 the owner sends an **open_call** request to the network with four connections. The first is assigned con_id 1, but the other three con_ids are *blank*. When the network responds to the owner with an **open_call** response, step 2, it includes in its response the connections listed in the same order they were sent by the client, each with a network generated con_id as shown. The network does not guarantee contiguous assignment of con_ids but will generally assign them in this manner. Given the dynamic nature of CMAP, where connections may be added or dropped by the owner or others, this contiguous assignment is not necessarily maintainable.

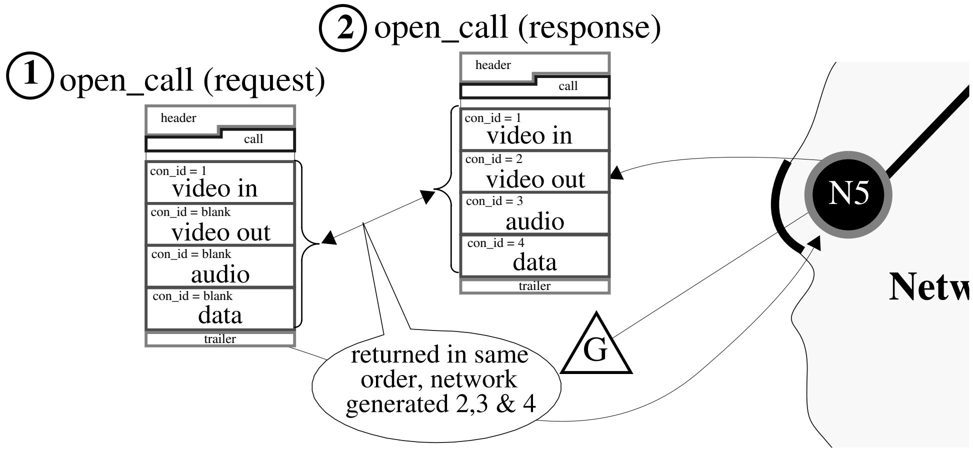


Figure 26. Mixing Client/Network Generated Connection Identifiers on open_call Negotiation.

6.4.3.2 ep_map negotiation in the add_ep operation

Figures 27 and 28 show the negotiation of the ep_id in an **add_ep** operation. In step 1 of Figure 27, client A sends an **add_ep** request to the network indicating that he wants to **invite** client F into a call (that currently has himself and client G). Client A requests that the default connection endpoint permission and the default connection endpoint mapping (ep_map) be used for the call's connection by leaving these fields blank. (Note that if client A were the call's owner, he could have chosen an override endpoint permission for client F.) The **add_ep** operation generates an **invite** operation to client F (step 2). The **invite** request lists the endpoint permission (*receive* and *transmit* allowed) and suggested endpoint mapping (*receive* and *transmit*) for client F. Figure 28 continues the example, showing client F responding to the **invite** (step 3) and requesting a mapping that is a valid subset of the permission, *transmitter with echo*. This negotiation is permitted within the offered endpoint permission and, after verifying that the network resources are available, the network acknowledges the negotiated connection mapping at client F with an **invite** confirmation (step 4), echoing the ep_map to client F. The network then completes the **add_ep** operation by acknowledging client A's request with an **add_ep** response in step 5. The response contains the endpoint mapping that client F renegotiated during the **invite** operation.

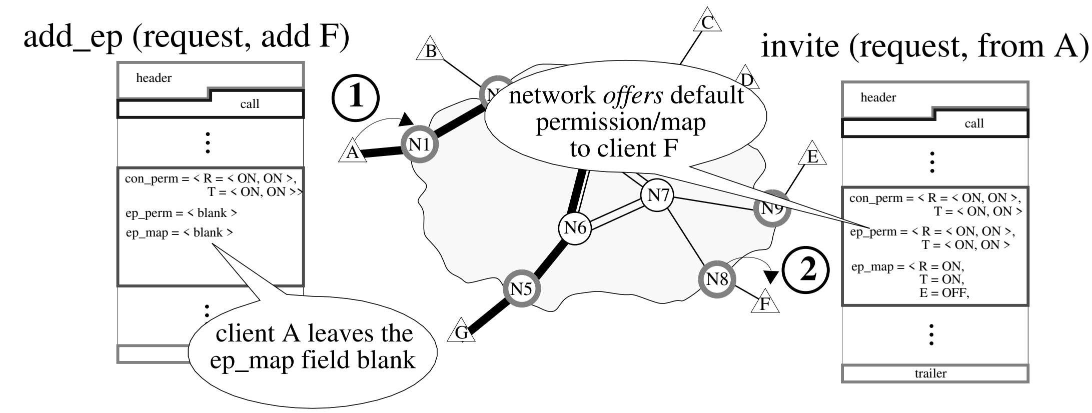


Figure 27. Non-Owner Client Inviting Another Client Using Default Permissions.

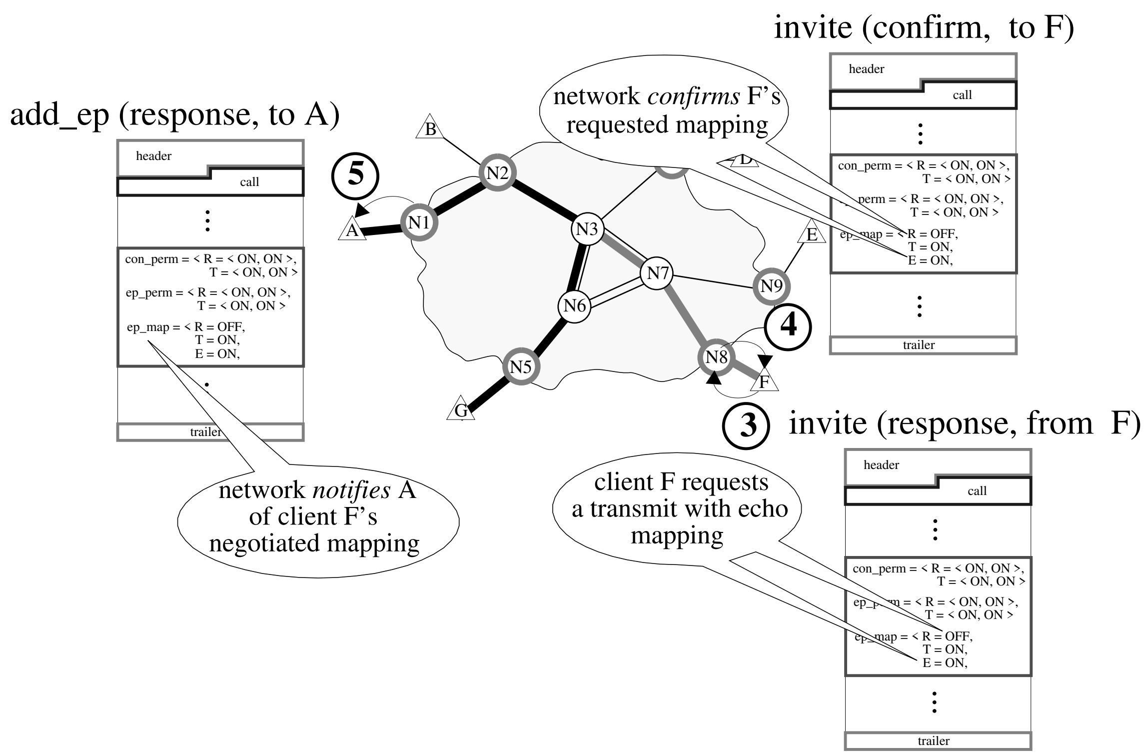


Figure 28. Invited Client Negotiating Connection Mapping (Continuation of Example from Figure 27).

6.4.3.3 ep_map negotiation in the open_con operation

Figures 29, 30 and 31 contain an example where call owner client A adds a new connection to an existing call and the two other endpoints in the call (clients G and F) each negotiate their respective endpoint mappings. Client G accepts the default connection mapping offered to it in the **open_con_invite** request, and client F renegotiates its offered endpoint mapping.

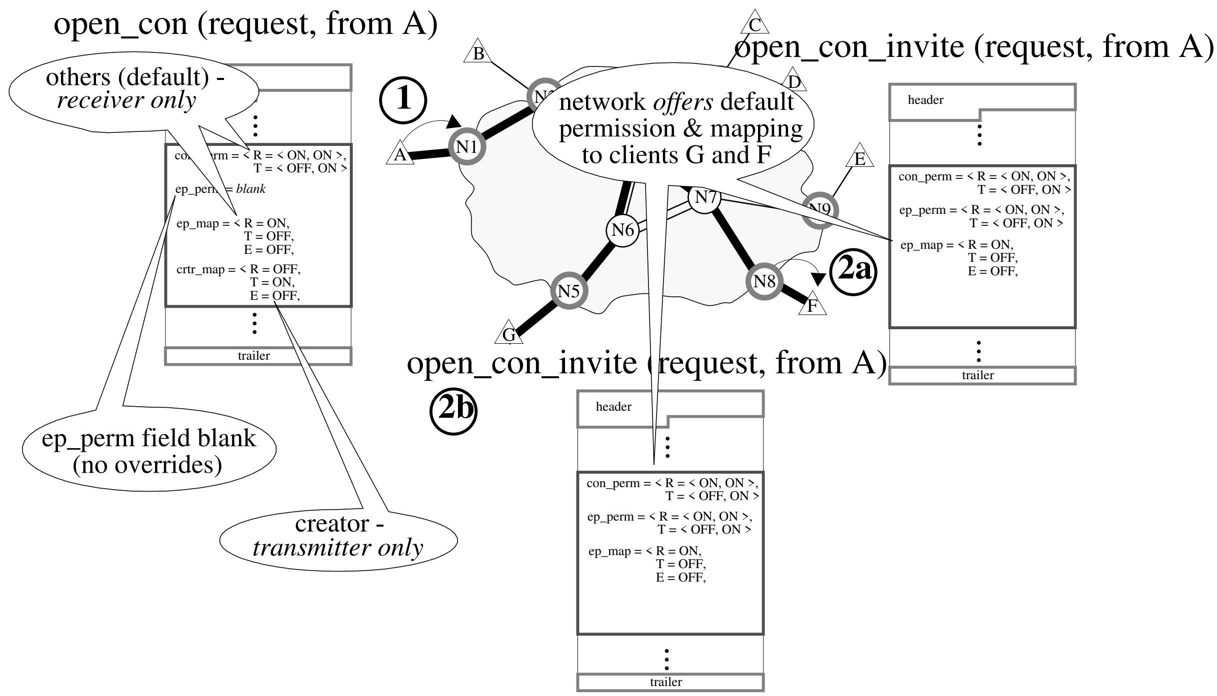


Figure 29. Owner Client Adding a Connection to a Call.

In step 1, Figure 29, client A issues the **open_con** request. The default connection permission and endpoint mapping specified by client A for this connection are:

permission: <receive = < perm = ON, changeability = ON>, transmit = < perm = OFF, changeability = ON>>

mapping: <receive = <ON>, transmit = <OFF>, echo = <OFF>> (*receive only*)

The mapping specifies that clients are typically connection *receivers* but may become *transmitters* if they choose (since changeability is ON). The creator also specified a creator endpoint mapping (crtr_map) to indicate how it intended to access the connection:

crtr_map: <receive = <OFF>, transmit = <ON>, echo = <OFF>> (*transmit only*)

The **open_con** request sent by A generates **open_con_invite** prompts to clients G and F, shown in steps 2a and 2b. The **open_con_invite** requests contain the offered endpoint permission and a suggested mapping.

Step 3a of Figure 30 shows client F responding to the **open_con_invite** and requesting a valid subset of the permission for a mapping of *transmitter/receiver* instead of the default *receiver only*. This negotiation is permitted within the offered endpoint permission and, after verifying that the network resources are available, the network acknowledges the negotiated connection mapping at client F with an **open_con_invite** confirmation in step 4a of Figure 31, echoing back the ep_map to client F. Step 3b of Figure 30 shows client G responding to the **open_con_invite** and agreeing to accept the default mapping of *receiver only*. This completes the ep_map negotiation for client G. The network completes the **open_con_invite** operation with client G by sending a confirmation in step 4b of Figure 31. The **open_con** operation completes in step 5 of Figure 31 when the network sends an **open_con** response to client A. Unlike the example of Section 6.4.3.2, where the initiating client (client A) was able to determine the added client F's negotiated connection mapping through the ep_map field that was contained in the **ad-**

d_ep response, the **open_con** response ep_map field has no context because the **open_con** operation spawned two **open_con_invites** and each negotiated a different mapping. Client A in this example may determine the mapping of Clients G and F through the **trace_ep** operation CMAP (not shown in the figures).

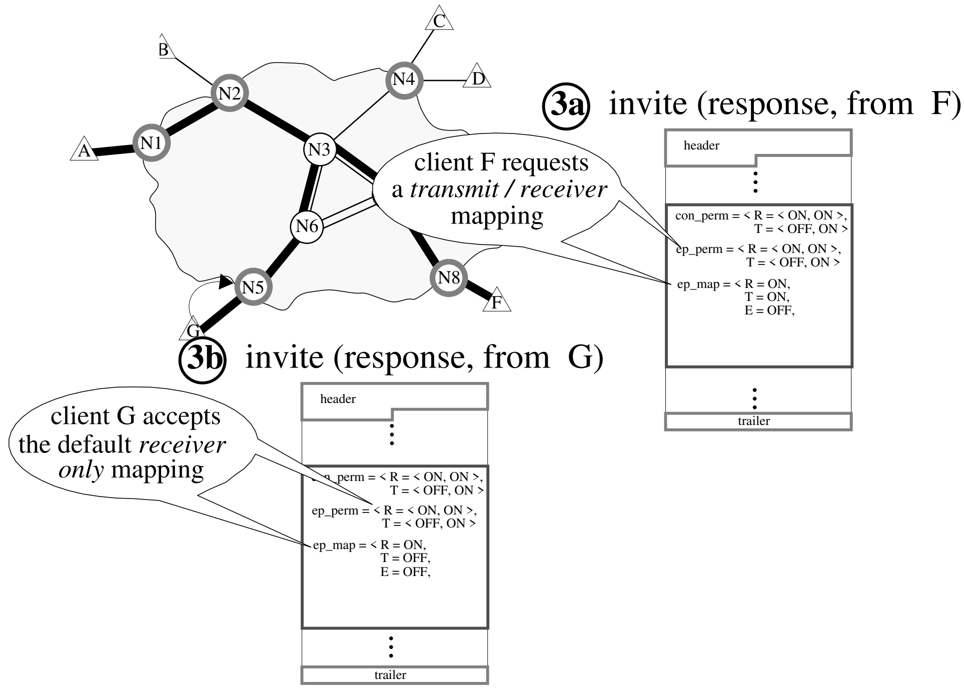


Figure 30. Invited Client Negotiating Connection Mapping (Continuation of Example from Figure 29).

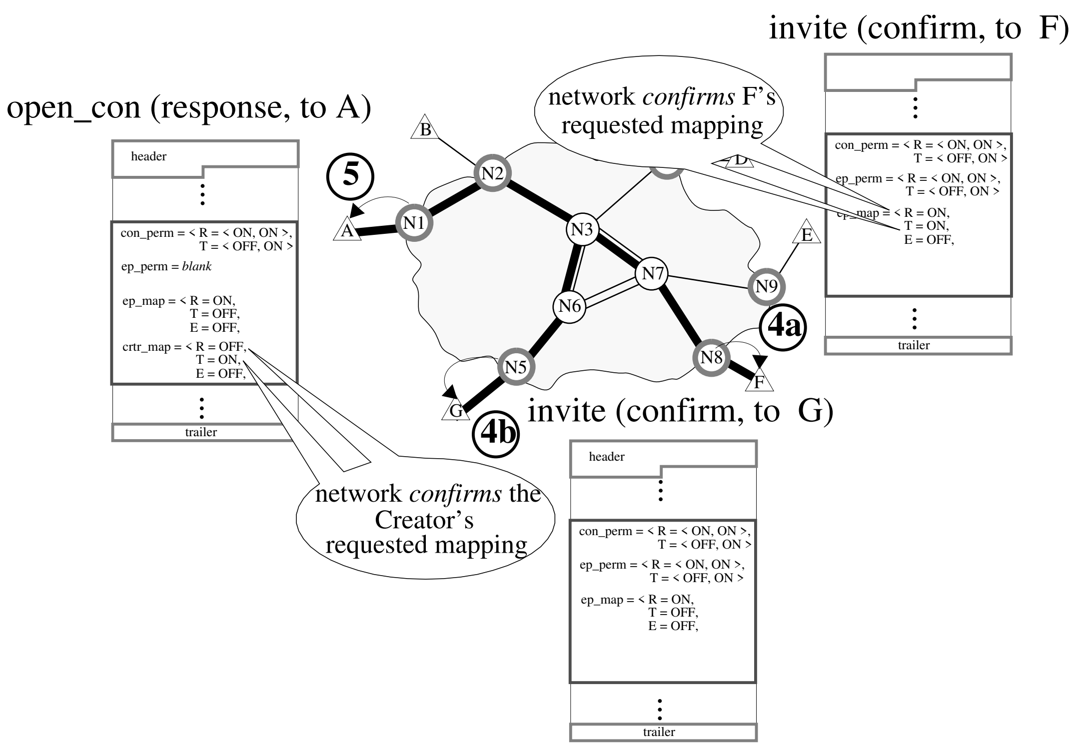


Figure 31. Invited Client Negotiating Connection Mapping (Continuation of Example from Figure 30).

7. CMAP Message Formats and Operational Effects

This section describes the CMAP message formats the detailed effects of the CMAP operations. Section 7.1 outlines the notation conventions used to diagram CMAP operations and discusses the bit and byte order of CMAP message transmission. Sections 7.2 and 7.3 describe the parameters passed in CMAP operations and the grouping of related parameters into CMAP *objects*. Finally, Section 7.4 gives the message layouts and details the operations themselves.

7.1 Notational Conventions and Message Transmission

Figure 32 shows a diagram of the **open_call** REQUEST. The example demonstrates how CMAP messages are presented in this document and the notational conventions used. CMAP message diagrams are laid out in eight-byte wide columns. Each field is represented by a space proportional to the size of the field. The name of the field appears in the space and the size of the field in bytes is in parenthesis next to the name.*

Related parameters within CMAP requests are grouped into *objects*. The **open_call** REQUEST shown in Figure 32 contains all six of the different types of objects defined (some of which appear multiple times and one of which is a component of a larger object). The six objects are: 1) the Header Object, 2) the Call Object, 3) the Endpoint Address Object, 4) the Connection Object, 5) the UNI Object (a component of the Connection Object), and 6) the Trailer Object.

The Header Object is made up of the *operation type (op_type), phase, message identifier (msg_id), call identifier (call_id), call status (call_status)* and two *reserved* fields. In the example the *op_type* and *phase* fields are filled in with the actual bit values for the **open_call** REQUEST.

The Call Object is made up of the *user call type*, *call type*, *accessibility (acc), monitoring (mon), modifiability*, *priority* and two reserved fields.

The Endpoint Address Object is made up of the *endpoint address (ep_addr)*, *endpoint* and two reserved fields.

The Connection Object is made up of the *connection identifier (con_id), connection type (con_type), notify flag (notify), user connection type (user_con_type), connection status (con_status)* and the *UNI Object*. Multiple connection objects may appear in one message, as indicated by the vertical dots.

The UNI Object is made up of the *connection permissions (con_perm), endpoint permissions (ep_perm), endpoint mapping (ep_map), creator mapping (crtr_map), transmit VPI (trans_vpi), transmit VCI (trans_vci), receive VPI (rcv_vpi), receive VCI (rcv_vci),* and two reserved fields.

Lastly, the Trailer Object is made up of an *options size* and *options* fields.

* There is one exception to this general rule: the call_id is separated into its component parts, the owner's address (o_addr) and the local call identifier (lcid), where each is shown separately.

In Figure 32 and throughout the rest of the document, the boundaries of a repeated set of fields are delineated by a heavy black line ▬ (for fields that are not a component of a larger object) or a heavy object line ▬ (for fields that are components of the objects define in Section 7.2).

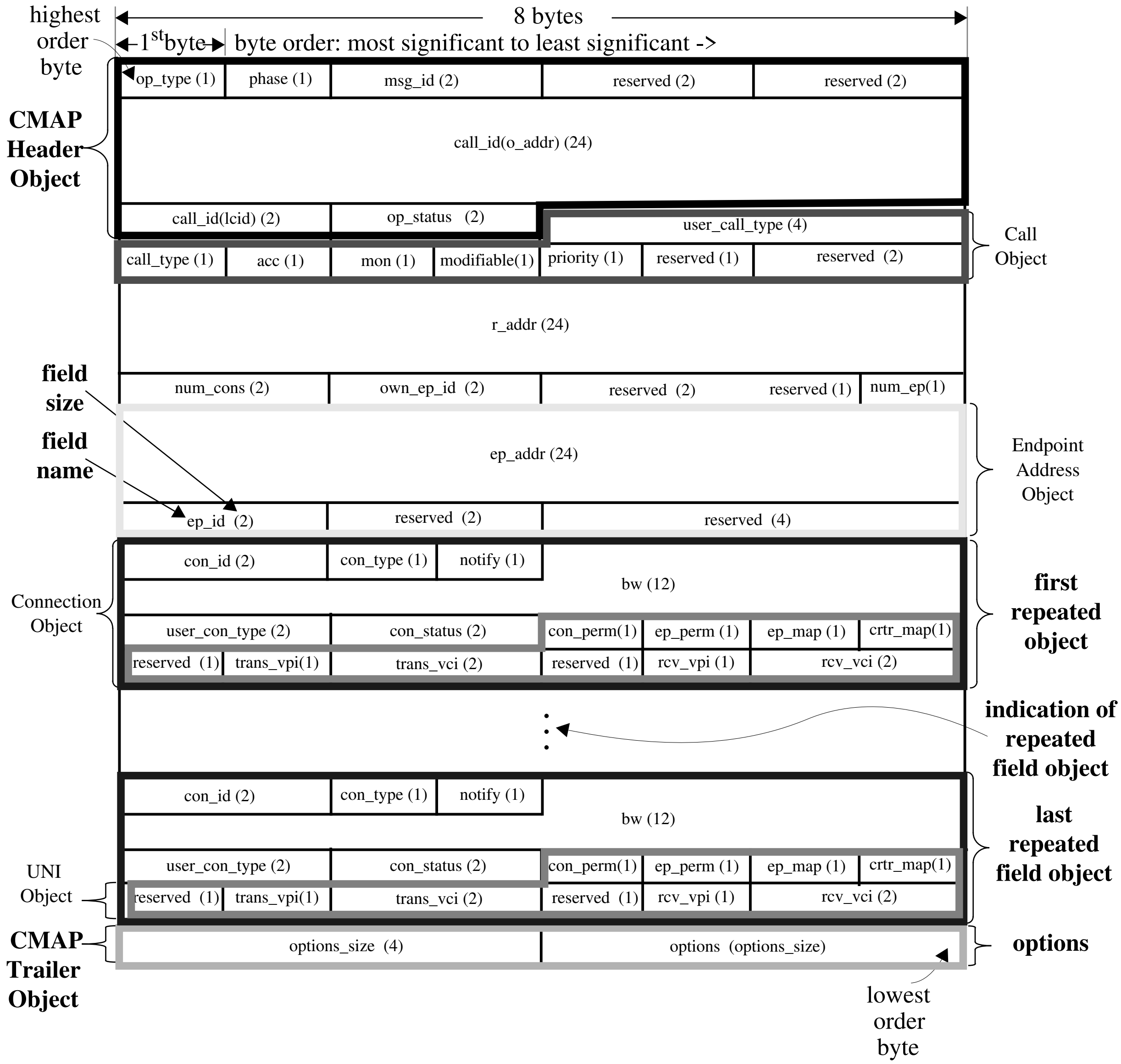


Figure 32. Common Message Format Characteristics (example open_call(REQUEST))

7.1.1 Common Rules for CMAP Messages

CMAP protocol data units follow the common rules listed below:

- The *op_type* field is the first field in every CMAP message, (for example, 00000001 for the **open_call** message shown in Figure 32).
- Every CMAP field is either 1 byte in length or an even number of bytes.

- A 1 byte field may start on any byte boundary.

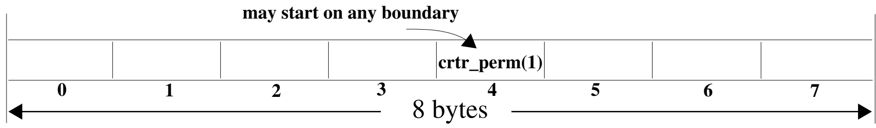


Figure 33. 1 Byte Field(s) Alignment

- A 2 byte field always starts on an even byte boundary.

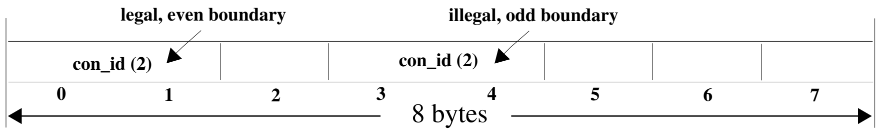


Figure 34. 2 Byte Field(s) Alignment

- A field of more than 2 bytes always starts on a 4-byte boundary.

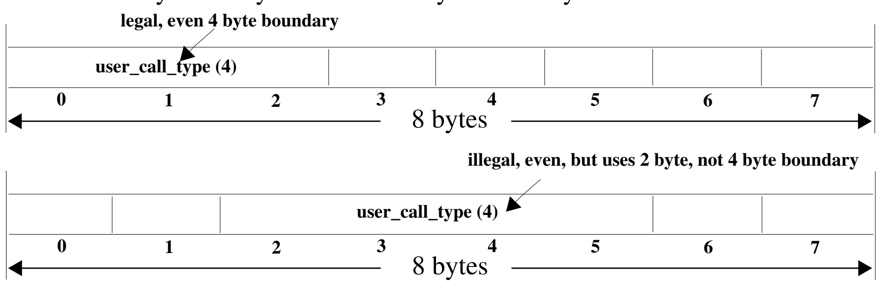


Figure 35. 3 or More Byte Field(s) Alignment

- A multiple byte field will be ordered from highest order byte to lowest order byte

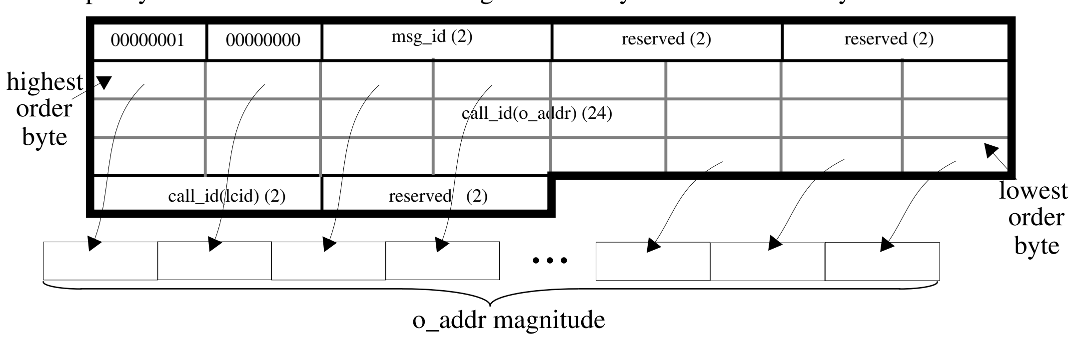


Figure 36. Byte Order for Multiple Byte Fields

7.1.2 CMAP Message Byte and Bit Order of Transmission

Figure 37 describes the CMAP message byte order of transmission. The CMAP message is always transmitted from the highest ordered byte to the lowest ordered byte. All CMAP bytes are bit ordered from the highest ordered bit to the lowest.

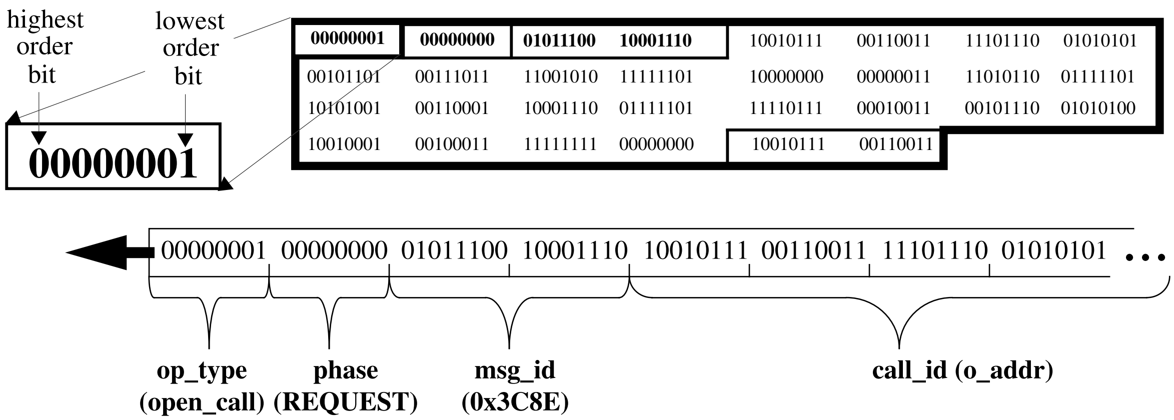


Figure 37. Byte/Bit Order of Transmission

7.2 Common CMAP Message Objects

There are five common objects that are used repeatedly in CMAP messages. The header and trailer objects are used by all CMAP messages. The call, connection and endpoint address objects are used by many messages. Each object groups a set of related parameters. Whenever they appear in a message they are always grouped in the same manner. This standard grouping helps to identify the object.

Some objects have fields that are specified as "unused (*n*)" during one or more phases of a CMAP operation. This means that the field has no context to the operation phase but may have context to a different phase of that operation or a different operation that utilizes information in the field. For example, the connection object has a *connection status* (con_status) field that the network uses to inform a client about the status of the connection in an operation. The con_status field is used, therefore, in a CMAP command response phase but is *unused* in a command's request phase. An unused field should not be confused with a reserved field. A reserved field is a field for which NO parameter is defined, whereas an unused field has a definition in some context in some other phase or operation.

7.2.1 Header Object

The common CMAP Header Object (Figure 38) is made up of the *operation type* (op_type), *phase* (phase), *message identifier* (msg_id), *call identifier* (call_id), *call status* (call_status) and two *reserved* fields. The call_id field is represented by its two sub-fields, *owner address* (o_addr) and *local call identifier* (lcid). The Header Object is distinguished in CMAP messages by a heavy line with a ▬ pen pattern.

CMAP messages are variable in length depending on the CMAP operation phase and options being used, each message, therefore, requires context specific interpretation. The op_type and phase fields tell the client or the network which operation template to use in interpreting the rest of the message.* The *message identification* (msg_id) field is assigned by the operation initiator (the client or network) and is used to associate the messages for the three CMAP phases (Request, Response, Confirm) of an operation. This allows parallel CMAP operations to be sent to the network (rather that having to serialize the requests). Without a unique *msg_id* the receiver would not be able to differentiate parallel operations, for example, two distinct add endpoint (**add_ep**) requests from a client.

The *call identifier* (call_id) field identifies the call that is being operated on. Most operations manipulate a specific call instantiation. The call_id field, however, is marked *unused* (24) on the **client_reset** and **network reset** operations because they refer to a global reset that affects all calls the client was participating in.

* Refer to each operation message definition for further information.

The final field of the Header Object is the *operation status* (op_status) field. The op_status field is used by the network to report the status of the call in the response and confirmation phases of an operation. Section 7.3.23 defines this field and the possible values that may be assigned to it are listed in Appendix C. This field is marked as *unused* whenever the object appears in a request operation.

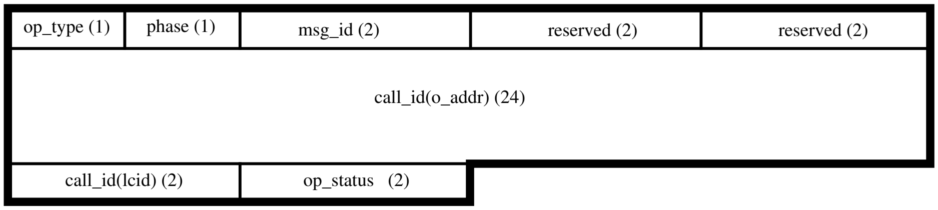


Figure 38. The CMAP Header Object

7.2.2 Trailer (Options) Object

All CMAP messages also have a common CMAP Trailer Object, also called the *options* object, Figure 39. The options object is provided to allow implementors to establish extensions to CMAP that may be put into a standard CMAP message. The object consists of a required field called *options size* (options_size) and an optional (if options_size = 0), variable length *options* field that contains the implementation dependent options. The Trailer Object is distinguished in CMAP messages by a heavy line with a pen pattern.

Figure 39. The CMAP Trailer Object

7.2.3 Call Object

The call object groups the parameters of a call into one object, Figure 40. The *call type* (call_type), *accessibility* (acc), *monitor* (mon), *modifiability* (modifiable) and *priority* (priority) fields comprise the parameters of a call described in Sections 3.1.1 through 3.1.5, respectively. The *user call type* (user_call_type) field is not interpreted by the network but is delivered end-to-end to clients whenever the Call Object is sent in a message. The Call Object is distinguished in CMAP messages by a heavy line with a pen pattern.

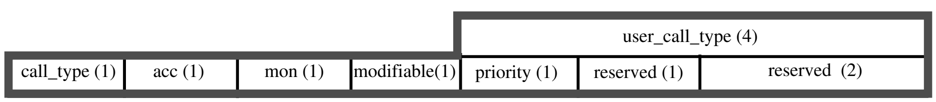


Figure 40. The CMAP Call Object

7.2.4 Connection Object

The connection object groups the parameters of a single connection, Figure 41. The *connection identifier* (con_id) is used to distinguish one connection from another in a multiconnection call. The *connection type* (con_type), *notification* (notify) and *bandwidth* (bw) fields comprise the connection parameters described in Sections 3.2.1, 3.2.5, and 3.2.2, respectively. The *user connection type* (user_con_type) field is not interpreted by the network (as with user_call_type) but is delivered end-to-end to clients whenever the Connection Object is sent in a message. The connection object is distinguished in CMAP messages by a heavy line with a pen pattern.

The connection object has a sub-object called the *UNI object* (uni_obj) that contains the connection default permissions, individual endpoint permission and mapping, the connection creator's mapping, and transmit and receive

VPI/VCI pairs for the instantiation of the connection at a particular client. This object is delineated in the connection object with a heavy line of the pen pattern to distinguish it from other fields.

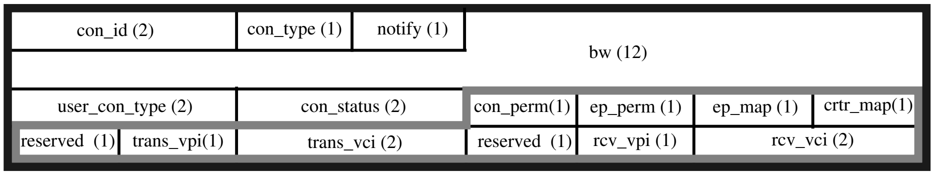


Figure 41. The CMAP Connection Object

7.2.5 Endpoint Address Object

The endpoint address object identifies a particular mapping of the call at a given client. It consists of *endpoint address* (ep_addr) and *endpoint identifier* (ep_id) fields, Figure 42. The ep_addr field identifies the client and the ep_id field identifies the instance (Section 6.3.4).

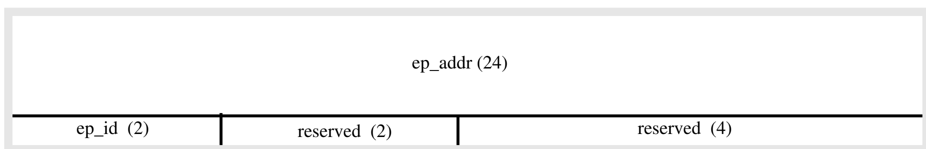


Figure 42. The Endpoint Address Object

7.3 Common CMAP Terms, Fields and Parameters

There are several fields or terms which are common to many CMAP messages. This section defines those parameters and terms so that the detailed descriptions do not have to be repeated for each operation.

7.3.1 reserved

A field is marked ***reserved*** if there is currently no defined use for this space in this message.

7.3.2 unused

A field is marked ***unused*** within an object if it is a defined field for the object but which has no meaning or application to the message being defined.

7.3.3 client addresses

The client address uniquely identifies network clients (Section 6.3). The address object is a concatenation of a four byte ***addr_type*** subfield and an ***address*** subfield, Figure 43. The currently defined ***addr_type*** field values are listed in Appendix B.

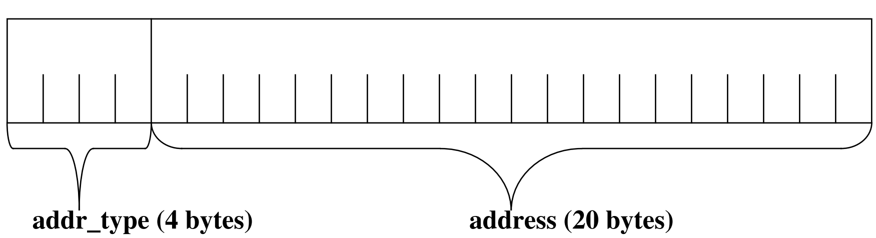


Figure 43. client address (24 bytes)

7.3.4 call identifier (call_id)

The ***call_id*** is a concatenation of the address of the call's owner, ***o_addr***, and a local call identifier, ***lcid*** (Section 6.3.2). The ***lcid*** must be generated by the client creating the call and is supplied in the **open_call REQUEST**. The network will verify that the ***lcid*** supplied in the **open_call REQUEST** is unique to the client address.

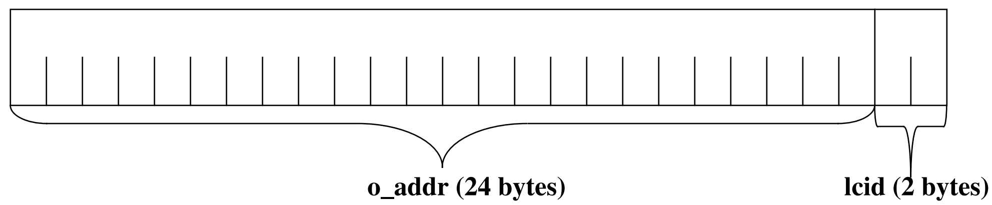


Figure 44. call_id (26 bytes)

7.3.5 message identifier (msg_id)

Since some CMAP operations spawn multiple operations (such as an **add_ep** operation spawning an **invite**), and since it is possible to have multiple operations of the same type from the same client occurring in parallel, it is necessary to be able to associate different phases of the same operation. For instance, if a client receives and open_call RESPONSE while it has two open_call REQUESTs active, it is important that it be able to determine to which REQUEST the RESPONSE applies. This is accomplished through a message identifier (***msg_id***). This identifier is generated by the initiating entity and must be unique to that entity at the time of creation.

A ***msg_id*** generated by a client must have the highest order bit set to 0. A ***msg_id*** generated by the network will have the highest order bit set to 1 (Figure 45).

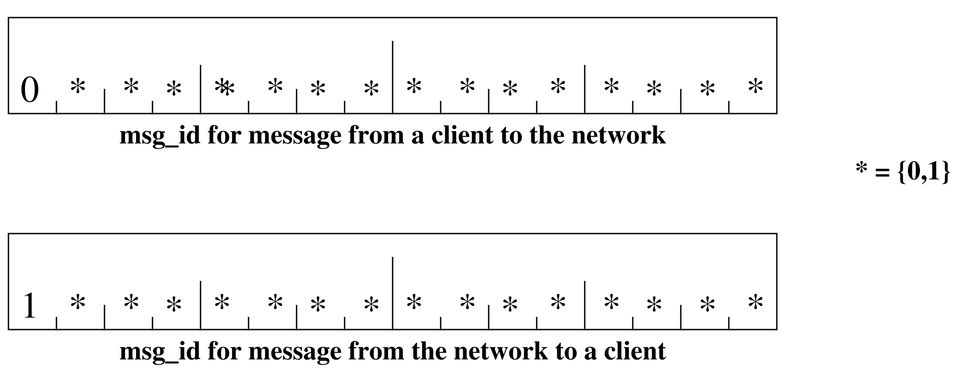


Figure 45. msg_id format

When a client generates a new **REQUEST** it will generate a new ***msg_id*** to go in it. When the network processes this **REQUEST** and generates a **RESPONSE** to send back to the client it will include this same ***msg_id*** in the **RESPONSE**. In this way, the client will be able to associate the **RESPONSE** with the correct **REQUEST**. The client must echo the network's ***msg_id*** back to the network in the same way when it receives a **REQUEST** from the network.

7.3.6 endpoint identifiers

Each participant within a call is identified by an *endpoint identifier* (Section 6.3.4). This identifier is generated by the network and is guaranteed to be unique for the client address within a given call. Three different endpoint identifiers appear in CMAP messages: ***ep_id***, ***add_ep_id*** and ***o_ep_id***. An ***ep_id*** is used in a **REQUEST** to identify the endpoint upon which an operation is to be performed on that endpoint. An ***add_ep_id*** specifies the endpoint identifier of a new endpoint in a **RESPONSE** to an operation that added that endpoint. An ***o_ep_id*** is used in the **RESPONSE** to an **open_call** to specify the endpoint identifier of the owner.

7.3.7 number of endpoints (num_ep)

In the **open_call REQUEST**, the ***num_ep*** field is used to indicate how many endpoints, other than the owner, are being added. The value in the ***num_ep*** field must be either 0 or 1.

In the **trace_call RESPONSE**, the ***num_ep*** field is used to indicate how many endpoints are being listed as participating in the call.

7.3.8 number of connections (num_cons)

The ***num_cons*** parameter is used to indicate how many connection specifications are being given in the CMAP message.

7.3.9 accessibility (*acc*)

The accessibility parameter, ***acc***, is defined by the owner of a call in the **open_call REQUEST**. This parameter determines who may add new endpoints to the call. If the value of ***acc*** is **CLOSED**, then only the owner may add new endpoints. If the value is **VERIFY**, then any client may attempt to add new a new endpoint to the call but the owner will be queried to verify the addition. If the value is **OPEN**, then any client may add a new endpoint to the call.

acc ∈ {OPEN, VERIFY, CLOSED}

7.3.10 monitor (mon)

The monitor parameter, ***mon***, is defined by the owner of a call in the **open_call REQUEST**. It determines if the owner, all transmitters, or every participant in the call should be notified when endpoints join, drop or are modified in the call.

mon = <owner ∈ {ON, OFF}, transmitters ∈ {ON, OFF}, all ∈ {ON, OFF} >

7.3.11 priority

The ***priority*** parameter specifies the priority level of the call. A value of **OVERRIDE** indicates that this call must be accepted and calls of lesser priority may be dropped in order to achieve this. A value of **PREEMPT** indicates that this call is of importance and calls with a priority of **NORMAL** should be dropped in order to achieve completion of this call.

priority ∈ {NORMAL, PREEMPT, OVERRIDE}

7.3.12 modifiability (modifiable)

The **modifiable** call parameter tells whether other endpoints in the call are able to add new connections to the call. If the **modifiable** parameter is set to **ON**, any endpoint in the call can add new connections. With it set to **OFF**, new connections can only be added by the owner.

modifiable ∈ {ON, OFF}

7.3.13 virtual path identifiers (VPI)

7.3.14 virtual circuit identifiers (VCI)

0,0 is *blank*.

7.3.15 connection permissions

Connection permissions are used to restrict the receive/transmit capabilities of endpoints in the connection. The default endpoint permission (***con_perm***) is assigned to endpoints unless overridden by the call's owner. The endpoint's override permission, ***ep_perm***, is set to the default until overridden. The bit layout of the ***con_perm*** and ***ep_perm*** fields is shown below (Figure 46).

$$\textbf{\textit{permissions}} = \langle \begin{array}{l} \text{receive} = \langle \text{perm} \in \{\text{ON, HOLD, OFF}\}, \text{changeability} \in \{\text{ON, VERIFY, OFF}\}\rangle, \\ \text{transmit} = \langle \text{perm} \in \{\text{ON, HOLD, OFF}\}, \text{changeability} \in \{\text{ON, VERIFY, OFF}\}\rangle\rangle \end{array}$$

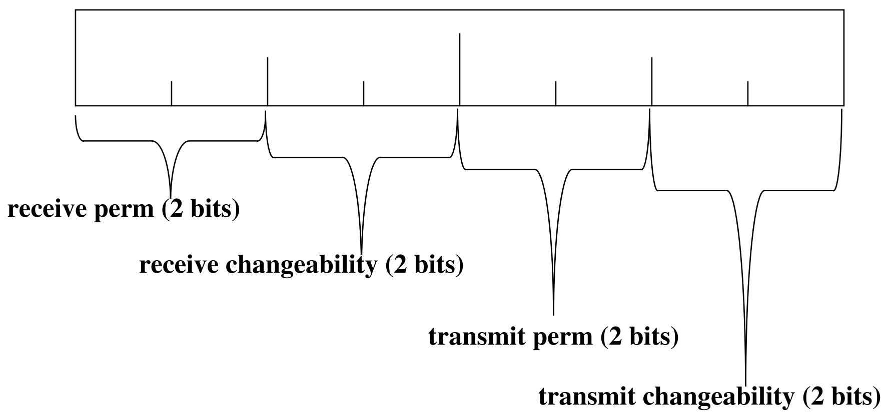


Figure 46. connection permissions (8 bits)

7.3.16 endpoint mapping

Each endpoint participating in a connection has a mapping for that connection which defines the level of that endpoint's participation. An endpoint mapping appears in CMAP messages as either an ***crtr_map*** to define the mapping for the creator of the connection or as an ***ep_map***, to define the mapping for a non-owner endpoint. The bit layout of the field is shown in Figure 47.

mapping = < receive = < perm ∈ {ON, HOLD, OFF}}>,
transmit = < perm ∈ {ON, HOLD, OFF}>,
echo = < perm ∈ {ON, OFF}> >

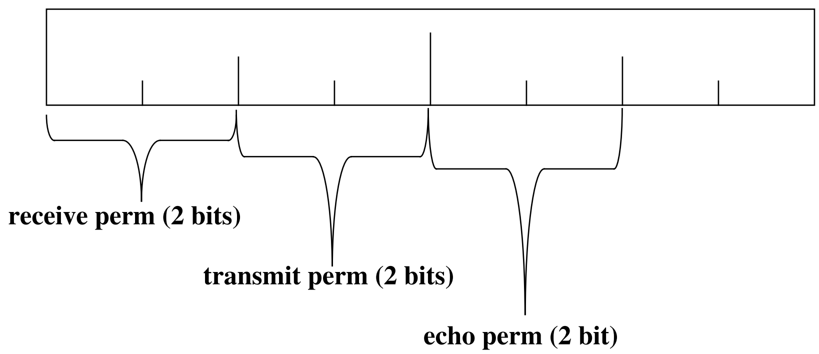


Figure 47. endpoint mapping (8 bits)

7.3.17 connection identifier (con_id)

Each connection is assigned a ***con_id***, unique to the call, at the time of the connection's creation. The ***con_id*** may be generated by the client creating the connection, in which case it is supplied in the **REQUEST** of the creating operation. The network will verify that the ***con_id*** supplied in the **REQUEST** is unique. If the client wants the network to generate the ***con_id***, the ***con_id*** field in the **REQUEST** should have every bit set to 1. In this case, the network will generate a ***con_id*** and return it to the client in the **RESPONSE**. The ***con_id*** is subsequently used to identify what connection is being operated on in any of the connection oriented operations.

7.3.18 call type (call_type)

The ***call_type*** parameter (Appendix B) is defined by the owner of the call. It designates whether the call is a multipoint call or is restricted to point-to-point. This parameter can help the network support a point-to-point call more efficiently, by possibly using internodal trunks. Modification of this call parameter during the life of a call may be difficult and costly.

7.3.19 connection type (con_type)

The ***con_type*** is a 3-tuple established by the creator of the connection. The first element specifies the type of ATM cell pipe that the connection requires and must be one of two values: *virtual path* (VP) or *virtual channel* (VC) (Appendix B). The second element specifies whether the connection will be **DYNAMIC** or **STATIC** (Appendix B). If a connection is **STATIC** then the bandwidth is fixed throughout the life of the call, which makes the connection more predictable, allowing more efficient use of network resources. **DYNAMIC** connections are the generic connection type, they can have their bandwidth modified during the life of the call. The last element specifies the *quality-of-service* (***qos***) desired and may have the values of **HIGH**, **MEDIUM** and **LOW** (Appendix B). The *quality-of-service* relates to options for cell loss behavior that may vary from network to network. The definition of the ***qos*** field, therefore, is left intentionally vague, see Section 3.2.1. If the network can implement different cell loss behavior strategies then the network control software will group these into the categories of **HIGH**, **MEDIUM** and **LOW**. Modification of this connection parameter during the life of a connection may be difficult and costly.

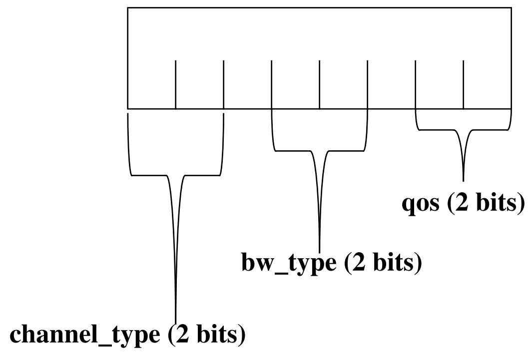


Figure 48. con_type (1 byte)

7.3.20 user call type (user_call_type)

The ***user_call_type*** field is available to communicate to potential endpoints some additional information about the type of call. For instance, the owner of a data call might want to indicate the intent of the data transfer (for example, FILE_TRANSFER, ELECTRONIC_MAIL, or IP_DATAGRAM). This field is delivered end-to-end without modification by the network.

7.3.21 user connection type (user_con_type)

The ***user_con_type*** field is available for use by the creator of a connection to communicate to potential endpoints some additional information about the type of connection. For instance, the creator of a video connection might want to specify what type of video format (for example, HDTV or NTSC) is being used. This field is delivered end-to-end without modification by the network.

7.3.22 bandwidth (bw)

A connection's bandwidth is a 3-tuple specified by the creator of the connection and consisting of ***peak***, ***average*** and ***peak_burst_length***. The ***peak*** and ***average*** parameters are expressed as integers representing cells per second. The ***peak_burst_length*** is measured in cells and indicates how many cells will be sent when the connection is sending a peak rate burst.

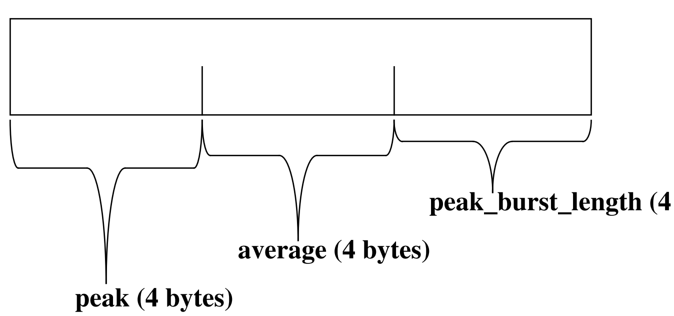


Figure 49. bw (12 bytes)

7.3.23 operation status (op_status)

The ***op_status*** field, returned in **RESPONSEs,** is divided into three subfields. The high order two bits are used to indicate whether there were errors in the call or connection specifications. If there were errors in the call specification, the highest order bit is set to 1 (**CALL_SPEC_ERROR**). If there were errors in any of the connection specifications the second highest order bit is set to 1 (**CON_SPEC_ERROR**). It is possible for both bits to be set in the same

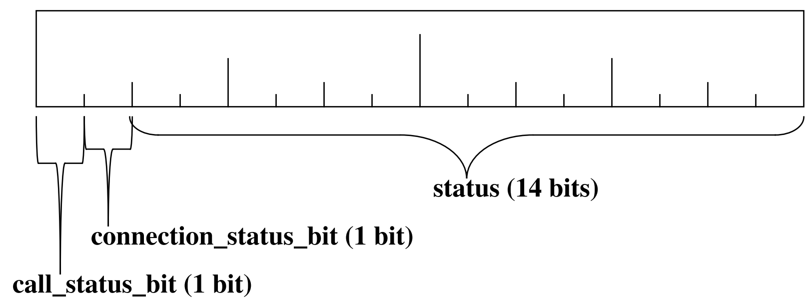


Figure 50. op_status (16 bits)

RESPONSE. If the ***call_status_bit*** is set(***call_status_bit*** = **CALL_SPEC_ERROR**), the client should check the ***status*** subfield (lowest 14 bits of ***op_status***) to determine the type of error that occurred. If the ***connection_status_bit*** is set(***connection_status_bit*** = **CON_SPEC_ERROR**), the client should check each of the ***con_status*** fields (Section 7.3.24) in the **RESPONSE** to determine which connection specification was in error.

A positive **RESPONSE**, one with both error bits set to 0 (***call_status_bit*** = **CALL_SPEC_OK**, ***connection_status_bit*** = **CON_SPEC_OK**, the rest of the ***op_status*** field should be set to **OK**), is referred to as an ACK. A negative **RESPONSE**, one with at least one of the error bits set, is referred to as a NACK.

Appendix B contains the values for all of these status fields and sub-fields, and Appendix C describes the error conditions.

7.3.24 connection status (con_status)

The ***con_status*** field will contain a value of OK if the corresponding connection specification was acceptable. If the connection specification contained an error that caused the **REQUEST** to fail, the ***con_status*** field will contain a value describing the error.

Appendix B contains the values for all of these status fields and sub-fields, and Appendix C describes the error conditions.

7.4 CMAP Operation Definitions

Following subsections define the individual CMAP operations. Each operation is defined using four sections: Synopsis, Data, Message Formats and Operation. The Synopsis section gives a very brief description of the operation. The Data section lists the individual fields used in the messages of the operation. A brief description of each field is given along with an enumerated list of predefined values, where applicable. For fields that have predefined values, these values are enumerated in Appendix B. An indication of whether the field is used in a message from the Client or from the Network is also given in the Data section. The Message Formats section defines the format for each message of the operation. The Operation section describes by prose and state diagrams how the Client CMAP Agent that sent or received the **REQUEST** should operate.

7.4.1 open_call

Synopsis:

This operation requests that a new call be created. An initial set of connections within that call are established as a part of a successful **open_call**. The requesting endpoint is set up as the owner of the call. One endpoint in addition to the owner may be added to the call as a part of the **open_call** operation. The **REQUEST** phase goes from owner to network.

Data:

- ***op_type*** - **open_call**
- ***phase*** - defines which type of message this is.

  phase ∈ {REQUEST, RESPONSE}
- ***msg_id*** - message identifier, unique to a client endpoint
- ***call_id*** **-** call identifier of the call to be opened
- ***user_call_type*** - user description of the type of this call. Not manipulated by the network, just passed end to end.
- ***call_type*** - specifies what type of call this is.

  call_type ∈ {POINT_TO_POINT, MULTIPOINT}
- ***acc*** - specifies whether endpoints may freely join the call.

  acc ∈ {OPEN, VERIFY, CLOSED}
- ***mon*** - specifies who should be notified when endpoints join, drop or are modified in the call.

  mon = <owner ∈ {ON, OFF}, transmitters ∈ {ON, OFF}, all ∈ {ON, OFF} >
- ***modifiability*** - specifies whether endpoints other than the owner may add connections to this call.

  modifiability ∈ {ON, OFF}
- ***priority*** - specifies a priority level for this call.

  priority ∈ {NORMAL, PREEMPT, OVERRIDE}
- ***num_cons*** - specifies the number of connections being specified in the **open_call**.
- ***own_ep_id*** - endpoint identifier for the owner of the call.
- ***add_ep_id*** - endpoint identifier for the endpoint being added, set to 0 if **num_ep** is 0.
- ***num_ep*** - specifies the number of endpoints being specified in the **open_call**, currently 0 or 1.
- ***ep_addr*** - if ***num_ep*** is 1, this field is present and it specifies the endpoint to be added, ep_addr is not present if **num_ep** is 0.
- ***op_status*** **-** describes the reason for the failure of the operation. (Section 7.3.23)

  op_status **= <*call_status_bit*** ∈ {CALL_SPEC_OK, CALL_SPEC_ERROR},
  connection_status_bit ∈ {CON_SPEC_OK, CON_SPEC_ERROR},
  status ∈ {OK, DUP_CALL_ID, DUP_LCID, BAD_CALL_ID_ADDR,
  INSUFF_BW, VERIFY_REFUSED,
  EP_REFUSED, DUP_MSG_ID, BAD_MSG_ID,
  BAD_PHASE_EXP_REQUEST,
  BAD_ACC, BAD_MON, BAD_CALL_TYPE,
  BAD_NUM_CONS, BAD_PRIORITY, BAD_MODIFIABLE,
  BAD_NUM_EP, BAD_EP_ADDR, TIMEOUT}>

- ***options_size*** - size of the following ***options*** field. If options_size is 0, then there is no ***options*** field present. The ***options_size*** field is always present.
- ***options*** - variable length field for introduction of new options and parameters. Currently there are no options defined.

For each connection specified in the **open_call REQ**, a connection specification is given which consists of the following information:

- ***con_id*** - connection identifier, uniquely identifies a connection within a call
- ***con_type*** - specifies what type of connection this is. This field is a 3-tuple.

    con_type = < channel_type ∈ {VP, VC},
    bw_ type ∈ {STATIC, DYNAMIC},
    qos ∈ {HIGH, MEDIUM, LOW}>

- ***con_perm*** - default permission specification for this connection

    con_perm = < receive = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}>,
    transmit = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}> >

- ***ep_perm*** - default endpoint permission specification for this connection.

    ep_perm = < receive = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}>,
    transmit = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}> >

- ***ep_map*** - specifies the UNI mapping to be used for the additional endpoint for this connection.

    ep_map = < receive = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}>,
    transmit = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}>,
    echo ∈ {ON, OFF} >

- ***crtr_map*** - specifies the UNI mapping to be used for the creator of this connection.

    crtr_map = < receive = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}>,
    transmit = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}>,
    echo ∈ {ON, OFF} >

- ***notify*** - specifies whether endpoints should be notified if this connection is changed via a **mod_con** operation (Section 7.4.13).

    notify ∈ {**YES**, **NO**}

- ***bw*** - bandwidth specification for the connection.

    bw = <peak, average, peak_burst_length>

- ***user_con_type*** - user description of the type of this connection. Not manipulated by the network, just passed end to end.
- ***trans_vpi*** - transmit virtual path identifier for the owner on this connection
- ***trans_vci*** - transmit virtual channel identifier for the owner on this connection.
- ***rcv_vpi*** - receive virtual path identifier for the owner on this connection
- ***rcv_vci*** - receive virtual channel identifier for the owner on this connection.
- ***con_status*** - this field is used to indicate what was potentially wrong with specific connection specifications. (Section 7.3.24)

    con_status ∈ {OK,
    DUP_CON_ID, BAD_CON_TYPE, BAD_NOTIFY,
    NO_AVAIL_VCI, RCV_VCI_IN_USE, TRANS_VCI_IN_USE,

RCV_VCI_RESERVED, RCV_VCI_NOT_SUPPORTED,
TRANS_VCI_RESERVED, TRANS_VCI_NOT_SUPPORTED,
NO_AVAIL_VPI, RCV_VPI_IN_USE, TRANS_VPI_IN_USE,
RCV_VPI_RESERVED, RCV_VPI_NOT_SUPPORTED,
TRANS_VPI_RESERVED, TRANS_VPI_NOT_SUPPORTED,
ILL_PERM_CHANGE,
BAD_CON_PERM, BAD_EP_PERM,
BAD_OWN_PERM, BAD_BW}

Message Formats:

open_call REQUEST

00000001	00000000	msg_id (2)		reserved (2)		reserved (2)	
call_id(o_addr) (24)							
call_id(lcid) (2)		unused (2)		user_call_type (4)			
call_type (1)	acc (1)	mon (1)	modifiable(1)	priority(1)	reserved (1)	reserved (2)	
r_addr (24)							
own_ep_id (2)		num_cons (2)		reserved (2)		reserved (1)	num_ep(1)
ep_addr (24)							
ep_id (2)		reserved (2)		reserved (4)			
con_id (2)		con_type (1)	notify (1)	bw (12)			
user_con_type (2)		unused (2)		con_perm(1)	ep_perm(1)	ep_map(1)	crtr_map(1)
reserved (1)	trans_vpi(1)	trans_vci (2)		reserved (1)	rcv_vpi (1)	rcv_vci (2)	
• • •							
con_id (2)		con_type (1)	notify (1)	bw (12)			
user_con_type (2)		unused (2)		con_perm(1)	ep_perm(1)	ep_map(1)	crtr_map(1)
reserved (1)	trans_vpi(1)	trans_vci (2)		reserved (1)	rcv_vpi (1)	rcv_vci (2)	
options_size (4)				options (options_size)			

Message Formats (continued):

open_call RESPONSE

00000001	**00000000**	**msg_id (2)**		**reserved (2)**		**reserved (2)**	
call_id(o_addr) (24)							
call_id(lcid) (2)		**op_status (2)**		**user_call_type (4)**			
call_type (1)	**acc (1)**	**mon (1)**	**modifiable(1)**	**priority(1)**	**reserved (1)**	**reserved (2)**	
own_ep_id (2)		**num_cons (2)**		**reserved (2)**		**reserved (1)**	**num_ep(1)**
ep_addr (24)							
ep_id (2)		**reserved (2)**		**reserved (4)**			
con_id (2)		**con_type (1)**	**notify (1)**	**bw (12)**			
user_con_type (2)		**con_status (2)**		**con_perm(1)**	**ep_perm(1)**	**ep_map(1)**	**crtr_map(1)**
reserved (1)	**trans_vpi(1)**	**trans_vci (2)**		**reserved (1)**	**rcv_vpi (1)**	**rcv_vci (2)**	
• • •							
con_id (2)		**con_type (1)**	**notify (1)**	**bw (12)**			
user_con_type (2)		**con_status (2)**		**con_perm(1)**	**ep_perm(1)**	**ep_map(1)**	**crtr_map(1)**
reserved (1)	**trans_vpi(1)**	**trans_vci (2)**		**reserved (1)**	**rcv_vpi (1)**	**rcv_vci (2)**	
options_size (4)				**options (options_size)**			

Operation:

A client sends an **open_call REQUEST** to the network to initiate a new call. Included in the **REQUEST** are a call identifier (**call_id**), a set of call parameters, a specification of an initial set of connections and an optional specification for an initial additional endpoint to be invited into the call. The **call_id** is made up of the initiating client's address and a local identifier (**lcid**), unique to the client. The initiating client has the option of either specifying the **lcid** or requesting the network to generate one for it. If the client generates the **lcid**, the network will verify its uniqueness. The initiating client also has the option of specifying the endpoint identifier for itself, **own_ep_id**, and the optional additional endpoint, **ep_id**. For more information on these negotiations see Section 6.4. The client initiating the **open_call REQUEST** is designated the owner of the call.

There are three ways in which the **open_call** operation can be used. The first and probably most useful is to create a call and add another endpoint in the same operation. This allows the creation of a complete point-to-point call between the owner and another endpoint in one operation. The second way to use the **open_call** is to create a call with no additional endpoint specified. Upon successful completion of this operation the owner will have a communications path to the network but will not be connected to any other endpoints. The third use of the **open_call** is a variation on the first. This is where the owner creates a call and specifies its own address for the endpoint to be added to the call.

The three methods of using the **open_call** listed in the above paragraph are discussed in more detail in the following paragraphs.

Case 1: Creating a Call and Adding Another Endpoint

In the first case, with regard to the call to be created, the Client CMAP Agent starts in the *no_call* state and generates an **open_call REQUEST**. This **REQUEST** specifies the connections to be included in the call and the additional endpoint to be added to the new call. The Client CMAP Agent sends the **REQUEST** and goes into the *call_pending* state. If there are any errors in the call or connection parameters or if there are any resources requested by the owner that are not available, the Client CMAP Agent will receive a negative **open_call RESPONSE**. The type of error will be reflected in the ***op_status***(Section 7.3.23) and ***con_status***(Section 7.3.24) fields in the **RESPONSE**. Upon receipt of a negative **RESPONSE**, the Client CMAP Agent should return to the *no_call* state, and if desired correct the errors and submit the **REQUEST** again.

If the owner did not give a complete specification for its VPI/VCI mapping for any of the connections listed in the call, the owner will receive an **invite REQUEST**, while in the *call_pending* state. Upon receiving the **REQUEST**, the Client CMAP Agent should make the transition to the *processing_invite* state. If the parameters suggested in the invite REQUEST are acceptable, the Client CMAP Agent should return a positive **invite RESPONSE**. If the parameters are not acceptable, the Client CMAP Agent may generate a new set of parameters and send them in an **invite RESPONSE** with the ***status*** subfield of the ***op_status*** field set to NEGOTIATING. The Client CMAP Agent should then make the transition to the *call_pending_invite_accepted* state. If the network accepts these new parameters, the owner will receive a positive invite CONFIRMATION and the Client CMAP Agent should make the transition to the *call_pending_invite_confirmed* state. Otherwise a negative **invite CONFIRMATION** will be received and the Client CMAP Agent should make the transition to the *call_pending_invite_confirmed* state.

If the parameters in the REQUEST were not acceptable to the owner the Client CMAP Agent may also send a negative **invite RESPONSE** and make the transition to the *invite_rejected* state. From this state the owner will receive a negative **open_call RESPONSE** and should make the transition to the *no_call* state.

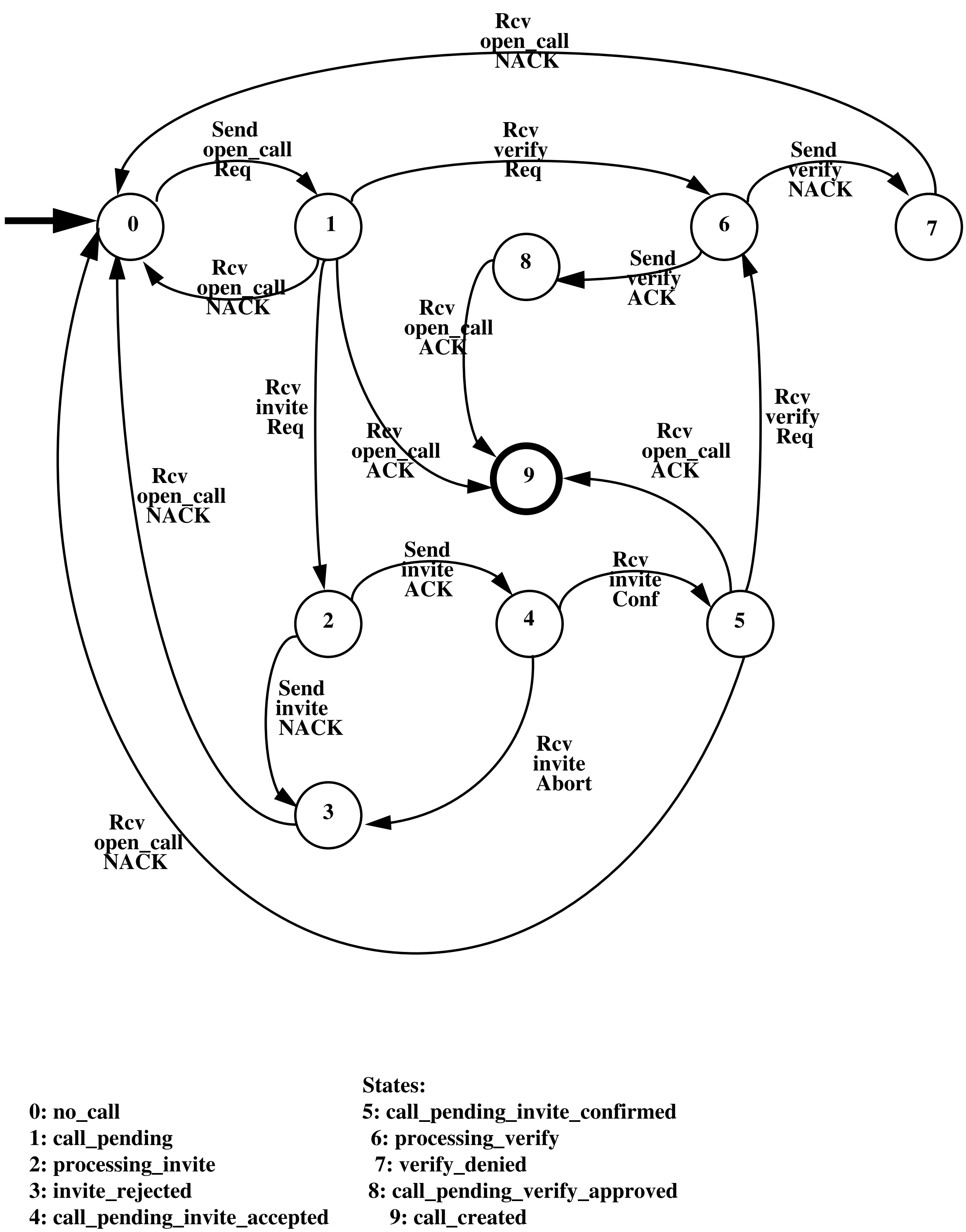
If the endpoint being added accepted all the connection permissions provided by the owner in the **open_call REQUEST** and the call was successfully set up in the network then the Client CMAP Agent will receive a positive **open_call RESPONS**E. Upon receipt of a positive **RESPONSE**, the Client CMAP Agent should make the transition to the *call_created* state.

If the endpoint being added only modified permissions for connections that had the changeability subfield set to ON and the call was successfully set up in the network then the Client CMAP Agent will receive a positive **open_call RESPONS**E. Upon receipt of a positive **RESPONSE**, the Client CMAP Agent should make the transition to the *call_created* state.

If the endpoint being added modified some of the connection permissions provided by the owner, and the ***changeability*** subfield of one or more of the modified permissions was set to **OFF** then the Client CMAP Agent of the owner will receive a negative **open_call RESPONSE** with the associated connection's ***con_status*** field (Section 7.3.24) set to **ILL_PERM_CHANGE**.

If the endpoint being added modified some of the connection permissions provided by the owner and the ***changeability*** subfield of one or more of the modified permissions was set to **VERIFY**, then the Client CMAP Agent of the owner will receive a **verify REQUEST**. Upon receipt of this **REQUEST**, the Client CMAP Agent should make the transition to the *processing_verify* state. This **REQUEST** should be processed and if the modified permissions are acceptable, a positive **verify RESPONSE** should be generated and sent to the network interface node. The Client CMAP Agent of the owner should then make the transition to the *call_pending_verify_approved* state. If the modified permissions were not acceptable, a negative **verify RESPONSE** should be generated and sent to the network interface node and the Client CMAP Agent should make the transition to the *verify_denied* state. From the *verify_denied* state, the Client CMAP Agent should receive a negative **open_call RESPONSE** with the status subfield of the ***op_status*** field(Section 7.3.23) set to **VERIFY_REFUSED**. The Client CMAP Agent should then make the transition to the *no_call* state. From the *call_pending_verify_approved* state, the Client CMAP Agent of the owner may receive either

Figure 51. Creating a call and adding another endpoint

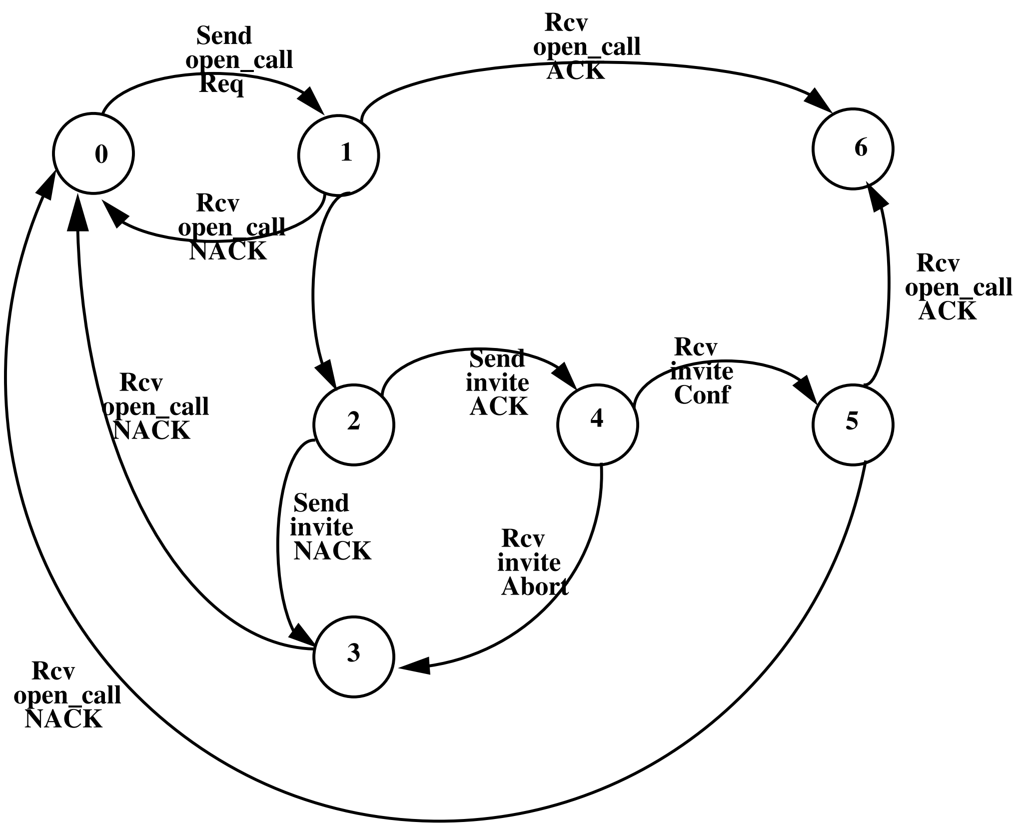


a positive or negative **open_call RESPONSE**. If the **RESPONSE** is positive, the Agent should make the transition to the *call_created* state. If the **RESPONSE** is negative, the Agent should make the transition to the *no_call* state.

Case 2: Creating a Call without Adding Another Endpoint

In the second case, with regard to the call to be created, the Client CMAP Agent starts in the *no_call* state and generates an **open_call REQUEST**. This **REQUEST** specifies the connections to be included in the call, but does

Figure 52. creating a call without adding another endpoint



States:
0: no_call
1: call_pending
2: processing_invite
3: invite_rejected
4: call_pending_invite_accepted
5: call_pending_invite_confirmed
6: call_created

not specify an additional endpoint. The Client CMAP Agent sends the **REQUEST** and goes into the *call_pending* state. If there are any errors in the call or connection parameters or if there are any resources requested by the owner that are not available, the Client CMAP Agent will receive a negative **open_call RESPONSE**. The type of error will be reflected in the ***op_status*** (Section 7.3.23) and ***con_status*** (Section 7.3.24) fields in the **RESPONSE**. Upon receipt of a negative **RESPONSE**, the Client CMAP Agent should return to the *no_call* state, and, if desired, correct the errors and submit the **REQUEST** again. If the **RESPONSE** is positive, the Agent should make the transition to the *call_created* state.

If the owner did not give a complete specification for its VPI/VCI mapping for any of the connections listed in the call, the owner will receive an **invite REQUEST**, while in the *call_pending* state. Upon receiving the **REQUEST**, the Client CMAP Agent should make the transition to the *processing_invite* state. If the parameters suggested in the invite REQUEST are acceptable, the Client CMAP Agent should return a positive **invite RESPONSE**. If the parameters are not acceptable the Client CMAP Agent may generate a new set of parameters and send them in an **invite RESPONSE** with the ***status*** subfield of the ***op_status*** field set to NEGOTIATING. The Client CMAP Agent should then make the transition to the *call_pending_invite_accepted* state. If the network accepts these new parameters, the owner will receive a positive invite CONFIRMATION and the Client CMAP Agent should make the transition to the *call_-pending_invite_confirmed* state. Otherwise a negative **invite CONFIRMATION** will be received and the Client CMAP Agent should make the transition to the *call_pending_invite_confirmed* state.

If the parameters in the REQUEST were not acceptable to the owner the Client CMAP Agent may also send a negative **invite RESPONSE** and make the transition to the *invite_rejected* state. From this state the owner will receive a negative **open_call RESPONSE** and should make the transition to the *no_call* state.

Case 3: Creating a Call and Adding Self as other Endpoint

In the third case, with regard to the call to be created, the Client CMAP Agent starts in the *no_call* state and generates an **open_call REQUEST**. This **REQUEST** specifies the connections to be included in the call and the additional endpoint (the owner) to be added to the new call. The Client CMAP Agent sends the **REQUEST** and goes into the *call_pending* state. If there are any errors in the call or connection parameters or if there are any resources requested by the owner that are not available, the Client CMAP Agent will receive a negative **open_call RESPONSE**. The type of error will be reflected in the ***op_status***(Section 7.3.23) and ***con_status***(Section 7.3.24) fields in the **RESPONSE**. Upon receipt of a negative **RESPONSE**, the Client CMAP Agent should return to the *no_call* state, and if desired correct the errors and submit the **REQUEST** again.

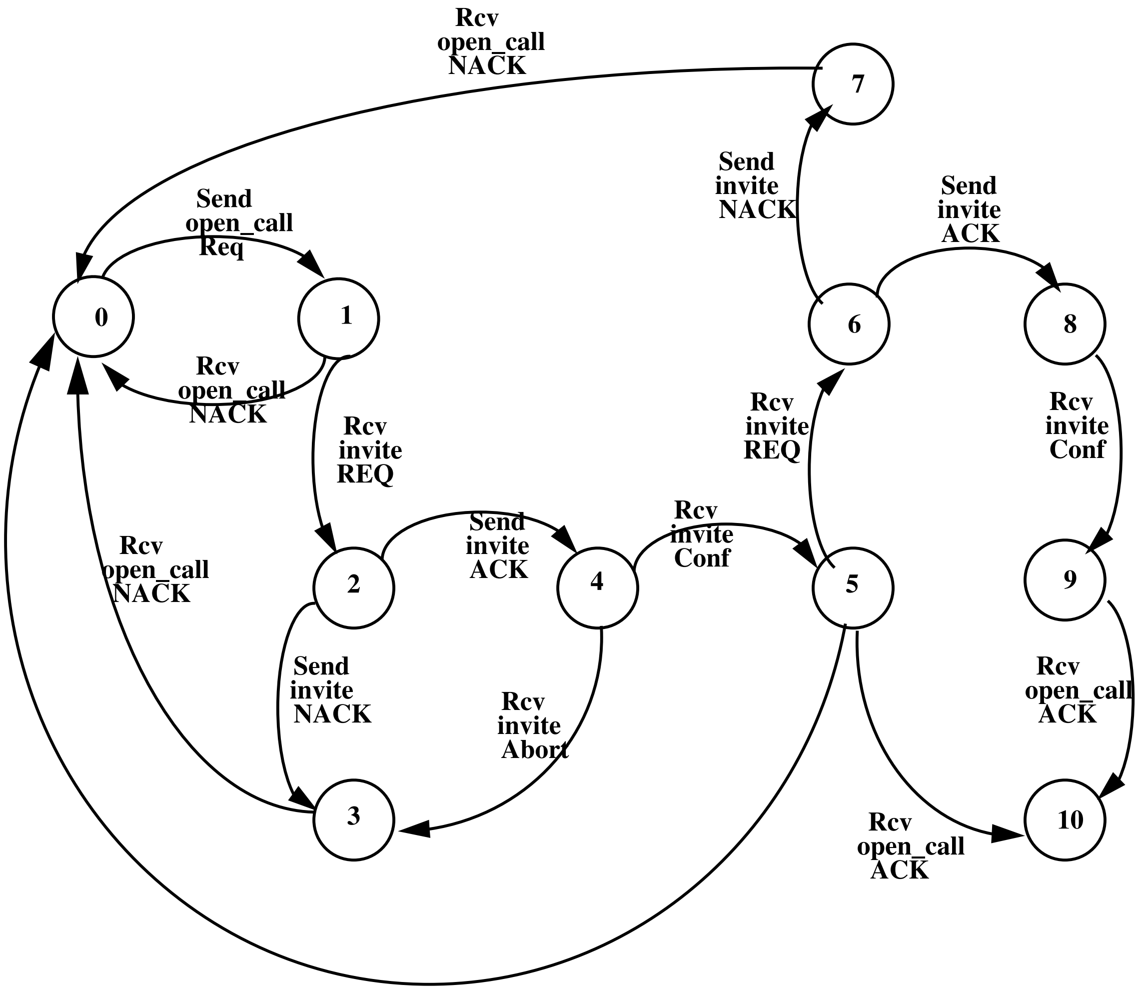
If the owner did not give a complete specification for its VPI/VCI mapping for any of the connections listed in the call, the owner will receive an **invite REQUEST**, while in the *call_pending* state. Upon receiving the **REQUEST**, the Client CMAP Agent should make the transition to the *processing_invite_1* state. If the parameters suggested in the invite REQUEST are acceptable, the Client CMAP Agent should return a positive **invite RESPONSE**. If the parameters are not acceptable the Client CMAP Agent may generate a new set of parameters and send them in an **invite RESPONSE** with the ***status*** subfield of the ***op_status*** field set to NEGOTIATING. The Client CMAP Agent should then make the transition to the *call_pending_invite_1_accepted* state. If the network accepts these new parameters, the owner will receive a positive invite CONFIRMATION, otherwise a negative **invite CONFIRMATION** will be received and the Client CMAP Agent should make the transition to the *call_pending_invite_1_confirmedxs* state.

If the parameters in the REQUEST were not acceptable to the owner the Client CMAP Agent may also send a negative **invite RESPONSE** and make the transition to the *invite_1_rejected* state. From this state the owner will receive a negative **open_call RESPONSE** and should make the transition to the *no_call* state.

The owner will always receive an **invite REQUEST** for itself being added as the other endpoint. If the owner had not given a full specification for its mapping, as described above, then this invitation for itself as the other endpoint would be the second invite. Otherwise, it would be the first. The description that follows assumes that it is the second invitation, but also contains state names in parenthesis as if it was the first.

Upon receiving the **invite REQUEST**, the Client CMAP Agent should make the transition to the *processing_invite_2* (*processing_invite_1*) state. If the parameters suggested in the invite REQUEST are acceptable, the Client CMAP Agent should return a positive **invite RESPONSE**. If the parameters are not acceptable the Client CMAP Agent may generate a new set of parameters and send them in an **invite RESPONSE** with the ***status*** subfield of the ***op_status*** field set to NEGOTIATING. The Client CMAP Agent should then make the transition to the *call_pending_invite_2_accepted* (*call_pending_invite_1_accepted*) state. If the network accepts these new parameters, the owner will receive a positive invite CONFIRMATION and the Client CMAP Agent should make the transition to the *call_-pending_invite_2_confirmed* (*call_pending_invite_1_confirmed*) state. Otherwise a negative **invite CONFIRMA-**

Figure 53. creating a call and adding self as other endpoint



States:

0: no_call
1: call_pending
2: processing_invite_1
3: invite_1_rejected
4: invite_1_accepted
5: invite_1_confirmed
6: processing_invite_2
7: invite_2_rejected
8: invite_2_accepted
9: invite_2_confirmed
10: call_created

TION will be received and the Client CMAP Agent should make the transition to the *call_pending_invite_2_confirmed* (*call_pending_invite_1_confirmed*) state.

From the *invite_2_rejected* (*invite_2_rejected*) state, the Client CMAP Agent should receive a negative **open_call RESPONSE** and make the transition to the *no_call* state. The reason for the negative **RESPONSE** will be reflected in the ***op_status***(Section 7.3.23) and ***con_status***(Section 7.3.24) fields in the **RESPONSE**.

From the *invite_2_confirmed* (*invite_2_confirmed*) state, the Client CMAP Agent should receive a positive **open_call RESPONSE** and make the transition to the *call_created* state.

7.4.2 close_call

Synopsis:

This operation requests that an existing call be terminated and may only be requested by the owner of the call. The **REQUEST** phase goes from owner to network.

Data:

- ***op_type*** - **open_call**
- ***phase*** - defines which type of message this is.

  phase ∈ {REQUEST, RESPONSE}
- ***msg_id*** - message identifier, unique to a client endpoint
- ***call_id*** **-** call identifier of the call to be opened
- ***op_status*** **-** describes the reason for the failure of the operation.(Section 7.3.23)

  op_status = <call_status_bit ∈ ***{***CALL_SPEC_***OK,*** CALL_SPEC_***ERROR},***
  connection_status_bit ∈ ***{***CON_SPEC_***OK},***
  status ∈ {OK, NOT_OWNER, UNKNOWN_CALL,
  BAD_MSG_ID, DUP_MSG_ID,TIMEOUT}>
- ***options_size*** - size of the following ***options*** field. If options_size is 0, then there is no ***options*** field present. The ***options_size*** field is always present.
- ***options*** - variable length field for introduction of new options and parameters. Currently there are no options defined.

Message Formats:

close_call REQUEST

00000010	**00000000**	**msg_id (2)**	**reserved (2)**	**reserved (2)**
call_id(o_addr) (24)				
call_id(lcid) (2)		**unused (2)**	**reserved (4)**	
options_size (4)			**options (options_size)**	

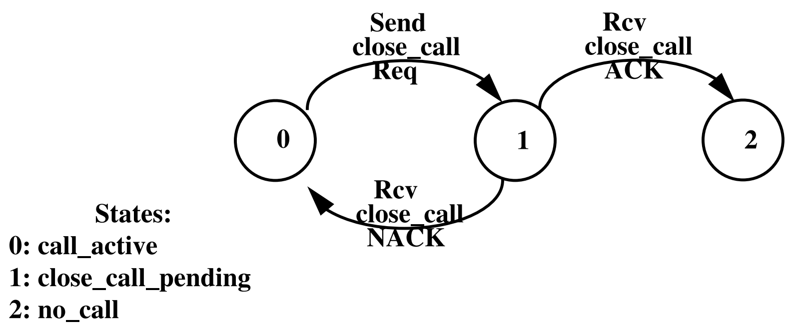
close_call RESPONSE

00000010	**00000001**	**msg_id (2)**	**reserved (2)**	**reserved (2)**
call_id(o_addr) (24)				
call_id(lcid) (2)		**op_status (2)**	**reserved (4)**	
options_size (4)			**options (options_size)**	

Operation:

The owner of a call sends a **close_call REQUEST** to the network to terminate the call. This **REQUEST** includes the ***call_id*** of the call to be terminated. The **close_call REQUEST** is propagated through the network along the paths

Figure 54. terminating a call



that the call uses. The endpoints that are taking part in the call receive **drop_from_call** messages. If the **REQUEST** is successful, an acknowledgment is sent back to the owner of the call with a ***op_status*** field of **OK**. If there is a failure in the network or at an endpoint, the **close_call** will still be acknowledged as successful. When the owner requests that a call be terminated, the call is terminated with respect to that client, provided a valid **close_call REQUEST** was issued. If the network has problems tearing down all the connections, then it is up to the connection managers to correct the situation (i.e. the client will not receive an error indication). The clients are not burdened by calls or connections that remain up after the **close_call REQUEST**. If the **close_call** is requested by an endpoint other than the owner of the call, a negative acknowledgment is returned with the status subfield of the ***op_status*** field(Section 7.3.23) of NOT_OWNER and the call remains active. If the ***op_status*** is returned as any other error value, the network interface node was not able to successfully process the **REQUEST** and the call remains in existence.

7.4.3 add_ep

Synopsis:

This operation requests that a new endpoint be added to an existing call. An **add_ep** may be requested by the owner of the call, the endpoint to be added, an endpoint already in the call or a client termination not part of the call. The **REQUEST** phase goes from client to network.

Data:

- ***op_type*** - **add_ep**
- ***phase*** - defines which type of message this is.

    phase ∈ {REQUEST, RESPONSE}
- ***msg_id*** - message identifier, unique to a client endpoint
- ***call_id*** - call identifier of the call to which the specified endpoint is to be added. This contains the address of the owner of the call and is used to determine if the owner is also the requester.
- ***ep_addr*** - address of endpoint to be added to the call
- ***ep_id*** - endpoint identifier for the endpoint added to the call.
- ***r_addr*** - address of endpoint requesting this addition. Needed so that endpoint being added can be told the full address, including the local extension.
- ***num_cons*** - specifies the number of connections being specified in the **add_ep**.
- ***op_status*** **-** describes the reason for the failure of the operation.(Section 7.3.23)

    op_status = <call_status_bit ∈ ***{***CALL_SPEC_***OK,*** CALL_SPEC_***ERROR},***
    connection_status_bit ∈ ***{***CON_SPEC_***OK,*** CON_SPEC_***ERROR},***
    status ∈ {OK, UNKNOWN_CALL, BAD_CALL_ID_ADDR,
    INSUFF_BW, VERIFY_REFUSED,
    EP_REFUSED, DUP_MSG_ID, BAD_MSG_ID,
    BAD_PHASE_EXP_REQUEST,
    BAD_NUM_CONS, BAD_EP_ADDR, TIMEOUT}>
- ***options_size*** - size of the following ***options*** field. If options_size is 0, then there is no ***options*** field present. The ***options_size*** field is always present.
- ***options*** - variable length field for introduction of new options and parameters. Currently there are no options defined.

For each connection specified in the **add_ep REQ**, a connection specification is given which consists of the following information:

- ***con_id*** - connection identifier, uniquely identifies a connection within a call
- ***con_type*** - specifies what type of connection this is. This field is a 3-tuple.

    con_type = < channel_type ∈ {VP, VC},
    bw_ type ∈ {STATIC, DYNAMIC},
    qos ∈ {HIGH, MEDIUM, LOW}>

- ***con_perm*** - default permission specification for this connection

    con_perm = < receive = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}>,

transmit = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}> >

- ***ep_perm*** - default endpoint permission specification for this connection.

  ep_perm = < receive = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}>,
  transmit = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}> >

- ***ep_map*** - specifies the UNI mapping to be used for the additional endpoint for this connection.

  ep_perm = < receive = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}>,
  transmit = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}>,
  echo ∈ {ON, OFF} >

- ***notify*** - specifies whether endpoints should be notified if this connection is changed via a **mod_con** operation (Section 7.4.13).

  notify ∈ {**YES**, **NO**}

- ***bw*** - bandwidth specification for the connection.

  bw = <peak, average, peak_burst_length>

- ***user_con_type*** - user description of the type of this connection. Not manipulated by the network, just passed end to end.
- ***trans_vpi*** - transmit virtual path identifier for the new endpoint on this connection
- ***trans_vci*** - transmit virtual channel identifier for the new endpoint on this connection.
- ***rcv_vpi*** - receive virtual path identifier for the new endpoint on this connection
- ***rcv_vci*** - receive virtual channel identifier for the new endpoint on this connection.
- ***con_status*** - this field is used to indicate what was potentially wrong with specific connection specifications. (Section 7.3.24)

  con_status ∈ {OK,
  DUP_CON_ID, BAD_CON_TYPE, BAD_NOTIFY,
  NO_AVAIL_VCI, RCV_VCI_IN_USE, TRANS_VCI_IN_USE,
  RCV_VCI_RESERVED, RCV_VCI_NOT_SUPPORTED,
  TRANS_VCI_RESERVED, TRANS_VCI_NOT_SUPPORTED,
  NO_AVAIL_VPI, RCV_VPI_IN_USE, TRANS_VPI_IN_USE,
  RCV_VPI_RESERVED, RCV_VPI_NOT_SUPPORTED,
  TRANS_VPI_RESERVED, TRANS_VPI_NOT_SUPPORTED,
  ILL_PERM_CHANGE, BAD_EP_PERM, BAD_BW}

Message Formats:

add_ep REQUEST

00000011	00000000	msg_id (2)		reserved (2)		reserved (2)	
call_id(o_addr) (24)							
call_id(lcid) (2)		unused (2)		reserved (2)		num_cons(2)	
r_addr (24)							
ep_addr (24)							
ep_id (2)		reserved (2)		reserved (4)			
con_id (2)		con_type (1)	notify (1)	bw (12)			
user_con_type (2)		unused (2)		con_perm(1)	ep_perm(1)	ep_map(1)	unused (1)
reserved (1)	trans_vpi(1)	trans_vci (2)		reserved (1)	rcv_vpi (1)	rcv_vci (2)	
⋮							
con_id (2)		con_type (1)	notify (1)	bw (12)			
user_con_type (2)		unused (2)		con_perm(1)	ep_perm(1)	ep_map(1)	unused (1)
reserved (1)	trans_vpi(1)	trans_vci (2)		reserved (1)	rcv_vpi (1)	rcv_vci (2)	
options_size (4)				options (options_size)			

Message Formats (continued):

add_ep RESPONSE

00000011	00000000	msg_id (2)		reserved (2)		reserved (2)	
call_id(o_addr) (24)							
call_id(lcid) (2)		op_status (2)		reserved (2)		num_cons(2)	
ep_addr (24)							
ep_id (2)		reserved (2)		reserved (4)			
con_id (2)		con_type (1)	notify (1)	bw (12)			
user_con_type (2)		con_status (2)		con_perm(1)	ep_perm(1)	ep_map(1)	unused (1)
reserved (1)	trans_vpi(1)	trans_vci (2)		reserved (1)	rcv_vpi (1)	rcv_vci (2)	
• • •							
con_id (2)		con_type (1)	notify (1)	bw (12)			
user_con_type (2)		con_status (2)		con_perm(1)	ep_perm(1)	ep_map(1)	unused (1)
reserved (1)	trans_vpi(1)	trans_vci (2)		reserved (1)	rcv_vpi (1)	rcv_vci (2)	
options_size (4)				options (options_size)			

Operation:

A client sends an **add_ep REQUEST** to the network to add a new endpoint to a call. Included in the **REQUEST** are the address of the endpoint to be added (***ep_addr***), the address of the client making the **REQUEST** (***r_addr***) and the identity of the call (***call_id***) to which the endpoint is to be added. The requesting client also has the option to specify the connections in which the new endpoint is to participate. There are three possible advantages for specifying the connections. The first is that the requesting client can specify the subset of the call's connections to which the new endpoint is to be added. The second advantage is that the requesting client can assign new endpoint permissions that differ from the default. The third advantage is that the **add_ep** operation may be completed faster when the requesting client specifies the connection specifications since the network interface will not have to search for the specifications if it is not already active in the call. The disadvantage to specifying the connections is that if any of the specifications are incorrect it could cause the **add_ep** operation to fail. If no connection specifications are given in the **add_ep REQUEST**, the operation will be performed using the default connection permissions as currently defined for the call.

The address of the requesting client, ***r_addr***, is included in the request because it may be of importance to both the endpoint being added and to the owner. The endpoint being added may require this information in deciding whether it wants to join the call. The owner may require this information in deciding whether to allow the addition to take place. Most of the information included in the ***r_addr*** field can be generated by the network interface node which receives the **add_ep REQUEST**. However, it is not able to generate the local extension portion of the client address. For this reason, the **add_ep REQUEST** must specify the ***r_addr*** field.

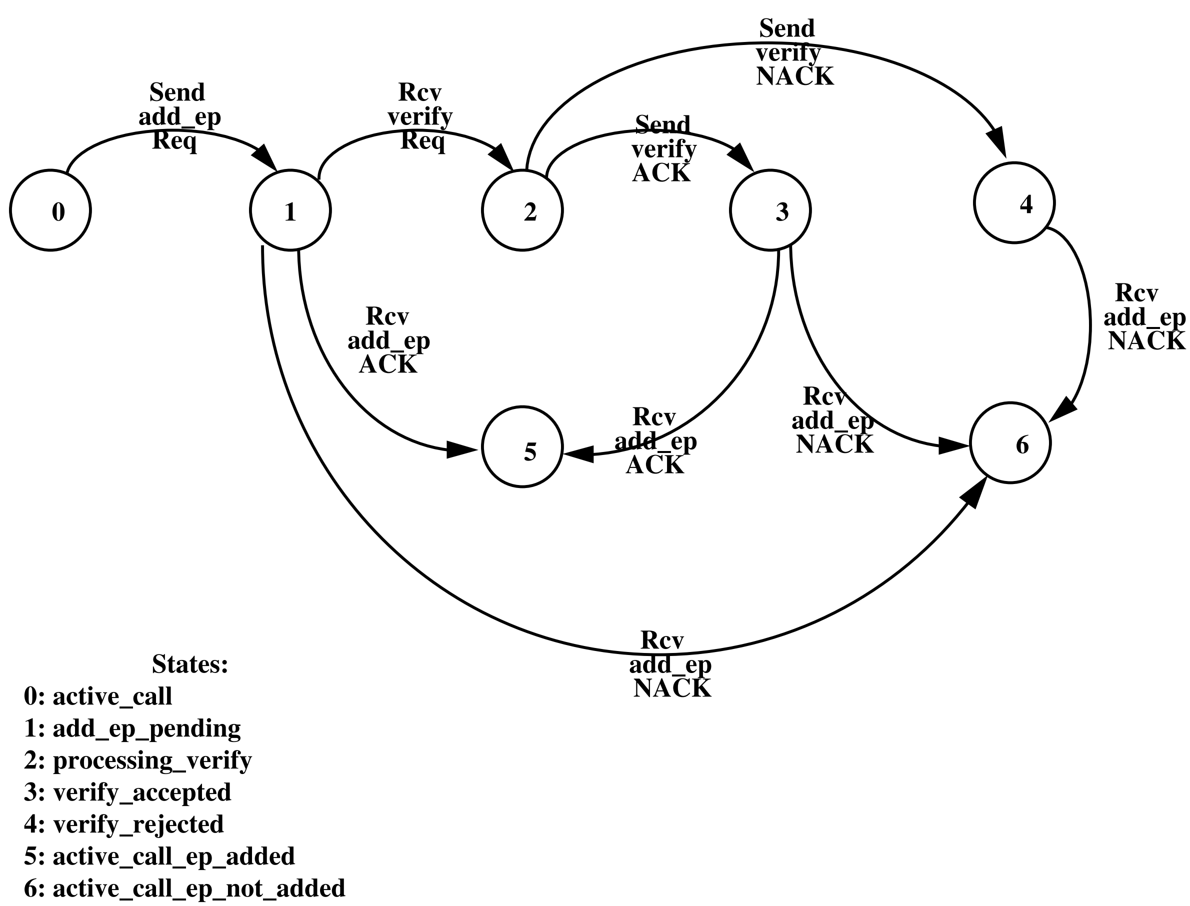
The requesting client has the option of specifying a suggested endpoint identifier, **ep_id**, for the endpoint being added (Section 6.4).

There are three ways in which that **add_ep** operation can be used. The first is for the owner to add another endpoint to the call. This would be the only use allowed in a call with the accessibility parameter, ***acc***, set to **CLOSED**. The second is for an endpoint to request to have itself added to a call. This method could be used, for example, to select a broadcast television channel. The third is for an endpoint other than the owner to request that another endpoint be added to the call. This use might be employed to add additional parties to a conference call. The requesting client need not be participating in the call to issue the **add_ep REQUEST**. The following paragraphs will discuss in more detail these three methods of using the **add_ep** operation.

Case 1: Owner Adding Another Endpoint

In this case, the owner is already active in the call so the Client CMAP Agent starts in the *active_call* state and generates an **add_ep REQUEST**. If the endpoint being added accepts all the connection permissions provided by the

Figure 55. Owner adding another endpoint



owner in the **add_ep REQUEST** and the endpoint is successfully connected to the call, then the Client CMAP Agent will receive an positive **add_ep RESPONSE**. Upon receipt of a positive **RESPONSE**, the Client CMAP Agent should make the transition to the *active_call_ep_added* state.

If the endpoint being added modified some of the connection permissions provided by the owner, and the ***change-ability*** subfield of one or more of the modified permissions was set to **OFF** then the Client CMAP Agent of the owner will receive a negative **add_ep RESPONSE** with the associated connection's ***con_status*** field (Section 7.3.24) set to **ILL_PERM_CHANGE**.

If the endpoint being added modified some of the connection permissions provided by the owner and the ***change-ability*** subfield of one or more of the modified permissions was set to **VERIFY**, then the Client CMAP Agent of the owner will receive a **verify REQUEST** (Section 7.4.7). Upon receipt of this **REQUEST**, the Client CMAP Agent should make the transition to the *processing_verify* state. This **REQUEST** should be processed and if the modified permissions are acceptable a positive **verify RESPONSE** should be generated and sent to the network interface node. The Client CMAP Agent of the owner should make the transition to the *verify_accepted* state. If the modified permissions were not acceptable then a negative **verify RESPONSE** should be generated and sent to the network interface node and the Client CMAP Agent should make the transition to the *verify_rejected* state. From the *verify_rejected* state, the Client CMAP Agent should receive a negative **add_ep RESPONSE** with the status subfield of the ***op_status*** field(Section 7.3.23) set to **VERIFY_REFUSED**. The Client CMAP Agent should then make the transition to the *active_call_ep_not_added* state. From the *verify_accepted* state, the Client CMAP Agent of the owner may receive either a positive or negative **add_ep RESPONSE**. If the **RESPONSE** is positive, the Agent should make the transition to the *active_call_ep_added* state. If the **RESPONSE** is negative, the Agent should make the transition to the *active_call_ep_not_added* state.

Case 2: Endpoint Adding Itself
In this case, the endpoint involved may or may not already be active in the call. The state diagram (Figure 56) for this case therefore has two possible starting states, *active_call* and *no_call*, for the Client CMAP Agent. From either state the Client CMAP Agent generates an **add_ep REQUEST**, sends it to its network interface node and makes the transition to the *add_ep_pending* state.

If the ***no_invite*** parameter was set to **ON** in the **add_ep REQUEST**, all of the call and connection specifications were acceptable and all of the requested resources were available, then the Client CMAP Agent will receive a positive **add_ep RESPONSE**. Upon receiving this **RESPONSE**, the Client CMAP Agent should make the transition to the *add_ep_succeeded* state.

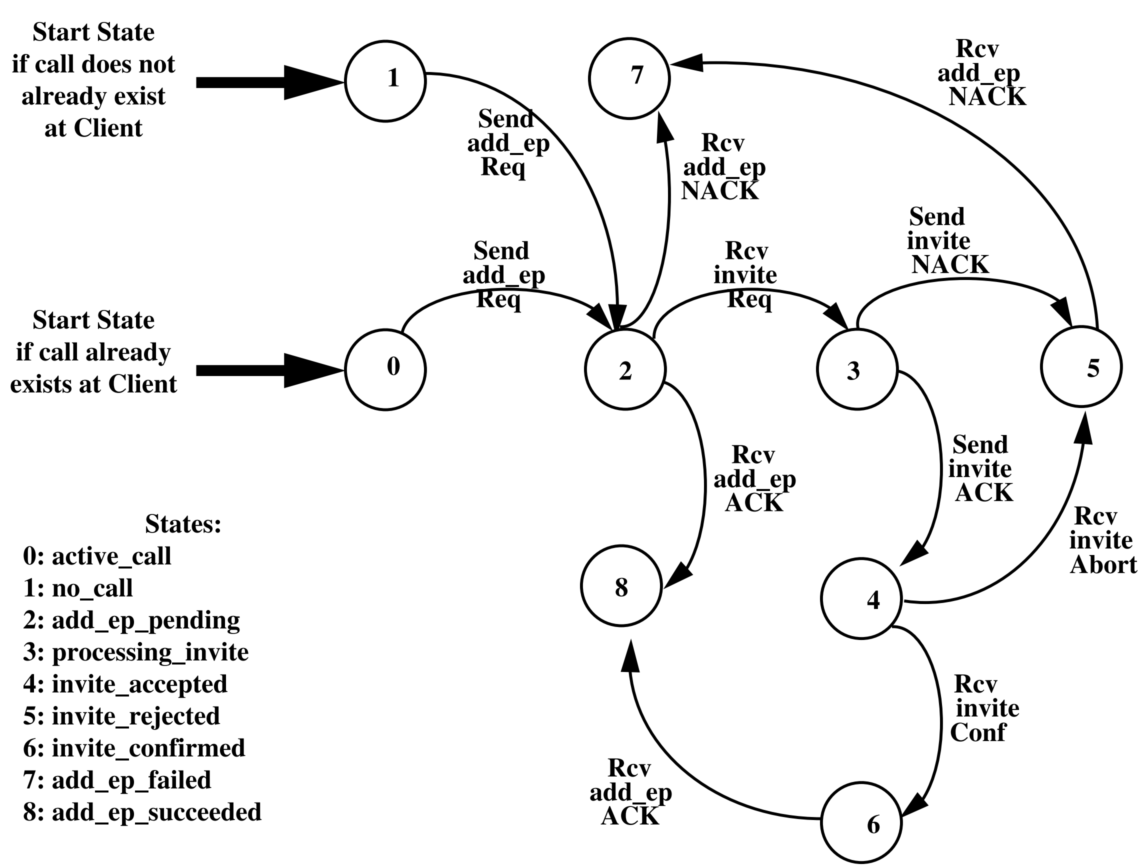
If the ***no_invite*** parameter was set to **OFF** in the **add_ep REQUEST**, all of the call and connection specifications were acceptable and all of the requested resources were available, then the Client CMAP Agent will receive an **invite REQUEST**. Upon receipt of this **REQUEST**, the Client CMAP Agent should make the transition to the *processing_invite* state. The **invite REQUEST** (Section 7.4.6) contains the parameters, including the VPI/VCI pair, for each of the connections to which this endpoint is being invited. If these parameters are acceptable, then the Client CMAP Agent should generate and send a positive **invite RESPONSE** and make the transition to the *invite_accepted* state. If the parameters are not acceptable, then the Client CMAP Agent should generate and send a negative **invite RESPONSE**, and make the transition to the *invite_rejected* state.

From the *invite_accepted* state the Client CMAP Agent may receive either a positive or negative **invite CONFIRMATION**. If a negative **CONFIRMATION** is received the Client CMAP Agent should make the transition to the *invite_rejected* state. If a positive **CONFIRMATION** is receive, the Agent should make the transition to the *invite_confirmed* state.

From the *invite_rejected* state, the Client CMAP Agent should receive a negative **add_ep RESPONSE** and make the transition to the *add_ep_failed* state. The reason for the negative **RESPONSE** will be reflected in the ***op_status***(Section 7.3.23) and ***con_status***(Section 7.3.24) fields in the **RESPONSE**.

From the *invite_confirmed* state, the Client CMAP Agent should receive a positive **add_ep RESPONSE** and make the transition to the *add_ep_succeeded* state.

Figure 56. Endpoint adding itself

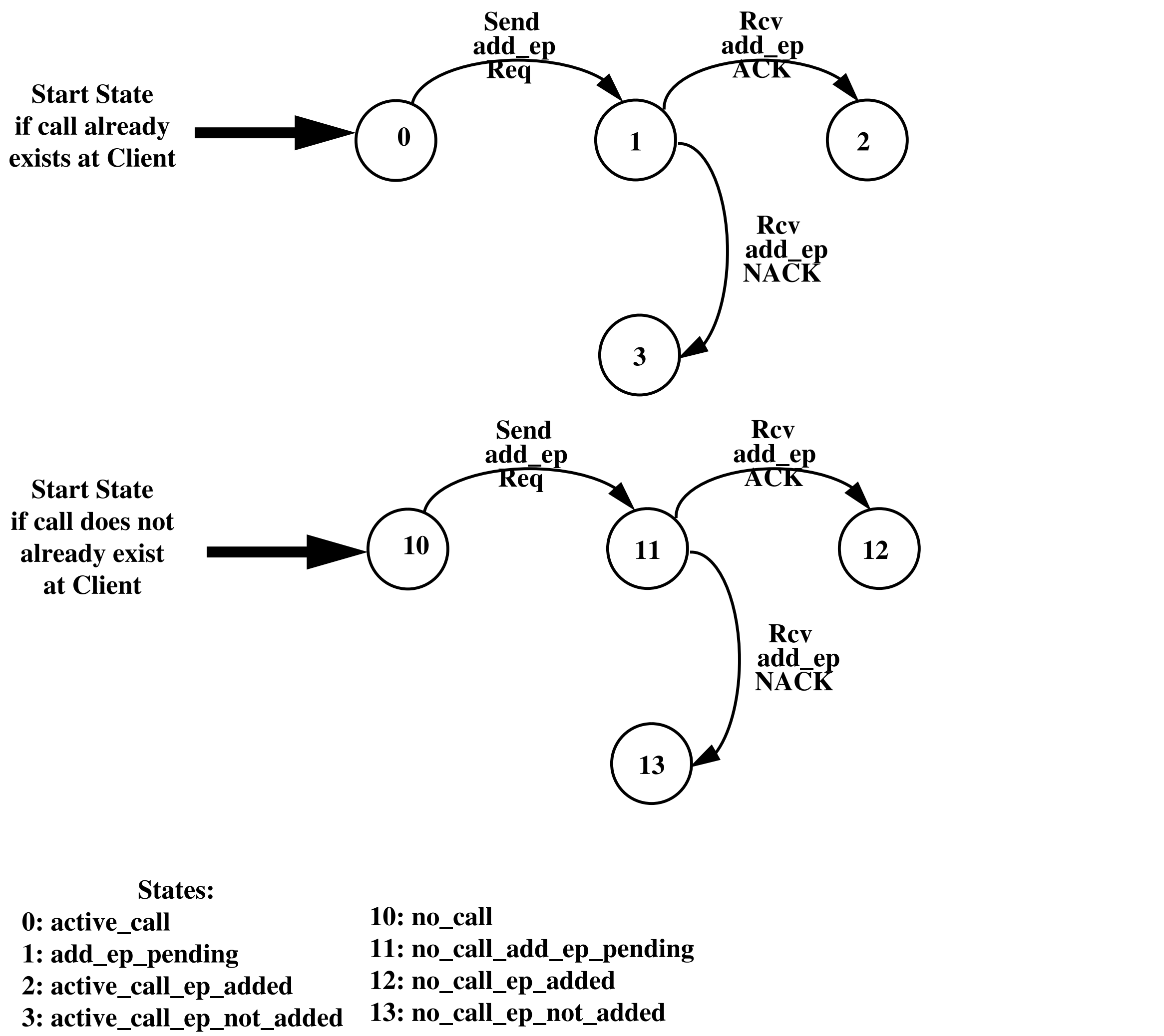


For an **add_ep** request coming from an endpoint other than the owner, if the endpoint is successfully added to the call and the **monitor** specification for the call is set to **ON**, then an **announce_ep** operation is triggered that informs the owner that a new endpoint has been added to the call.

Case 3: Endpoint Adding Itself

In this case, the endpoint making the request may or may not already be active in the call. The state diagram (Figure 57) for this case therefore has two possible starting states, *active_call* and *no_call*, for the Client CMAP Agent. From the *active_call* (*no_call*) state, the Client CMAP Agent generates an **add_ep REQUEST**, sends it to its network interface node and makes the transition to the *add_ep_pending* (*no_call_add_ep_pending*) state. From this state the Client CMAP Agent will receive either a positive or negative **add_ep RESPONSE**. If it is a positive **RESPONSE** it should make the transition to the *active_call_ep_added* (*no_call_ep_added*) state. If the **RESPONSE** was negative, it should make the transition to the *active_call_ep_not_added* (*no_call_ep_not_added*) state. The reason for the negative **RESPONSE** will be reflected in the ***op_status***(Section 7.3.23) and ***con_status***(Section 7.3.24) fields in the **RESPONSE**.

Figure 57. Non-owner Endpoint adding another endpoint



For an **add_ep** request coming from an endpoint other than the owner, if the endpoint is successfully added to the call and the **monitor** specification for the call is set to **ON**, then an **announce_ep** operation is triggered that informs the owner that a new endpoint has been added to the call.

7.4.4 drop_EP

Synopsis:

This operation requests that an endpoint be dropped from a call. This may be requested by the owner of the call or the endpoint to be dropped. The REQUEST phase goes from client to network.

Data:

- ***op_type*** - **drop_ep**
- ***phase*** - defines which type of message this is.

  phase ∈ {REQUEST, RESPONSE}
- ***msg_id*** - message identifier, unique to a client endpoint
- ***call_id*** - call identifier of the call from which the specified endpoint is to be dropped. This contains the address of the owner of the call and is used to determine if the owner is also the requester.
- ***op_status* -** describes the reason for the failure of the operation.(Section 7.3.23)

  op_status* = <*call_status_bit ∈ {CALL_SPEC_OK, CALL_SPEC_ERROR},

  connection_status_bit ∈ {CON_SPEC_OK, CON_SPEC_ERROR},

  status ∈ {OK, NO_SUCH_EP, ILL_DROP_EP_REQUEST, TIMEOUT}
- ***ep_addr*** - address of endpoint to be dropped from the call
- ***ep_id*** - identifier of the endpoint being dropped.
- ***r_addr*** - address of endpoint requesting the drop
- ***options_size*** - size of the following ***options*** field. If options_size is 0, then there is no ***options*** field present. The ***options_size*** field is always present.
- ***options*** - variable length field for introduction of new options and parameters. Currently there are no options defined.

Message Formats:

drop_ep REQUEST

00000100	00000000	msg_id (2)	reserved (2)	reserved (2)
call_id(o_addr) (24)				
call_id(lcid) (2)		unused (2)	reserved (4)	
r_addr (24)				
ep_addr (24)				
ep_id (2)		reserved (2)	reserved (4)	
options_size (4)			options (options_size)	

drop_ep RESPONSE

00000100	00000001	msg_id (2)	reserved (2)	reserved (2)
call_id(o_addr) (24)				
call_id(lcid) (2)		op_status (2)	reserved (4)	
options_size (4)			options (options_size)	

Operation:

The **drop_EP REQUEST** may come from the owner of the call or the endpoint to be dropped. The ***r_addr*** field specifies the address of the endpoint requesting the drop and the ***ep_addr*** specifies which endpoint that is being dropped. The address of the endpoint that made the request may be of importance to the endpoint being dropped. Most of the information included in the ***r_addr*** field could be generated by the network interface node that receives the **drop_EP** request. However, the network interface node could not generate the local extension portion of the ***r_addr*** field. For this reason, the **drop_EP** request must contain the ***r_addr*** field. If the request comes from the owner, a **drop_from_call** operation is triggered. The **drop_from_call** is sent to the endpoint that it is being dropped and includes the address of the requester, ***r_addr***.

If an endpoint requests that it be dropped, it is dropped immediately and the owner of the call is informed of this if the monitor endpoint specification (***mon***) is set to TRUE.

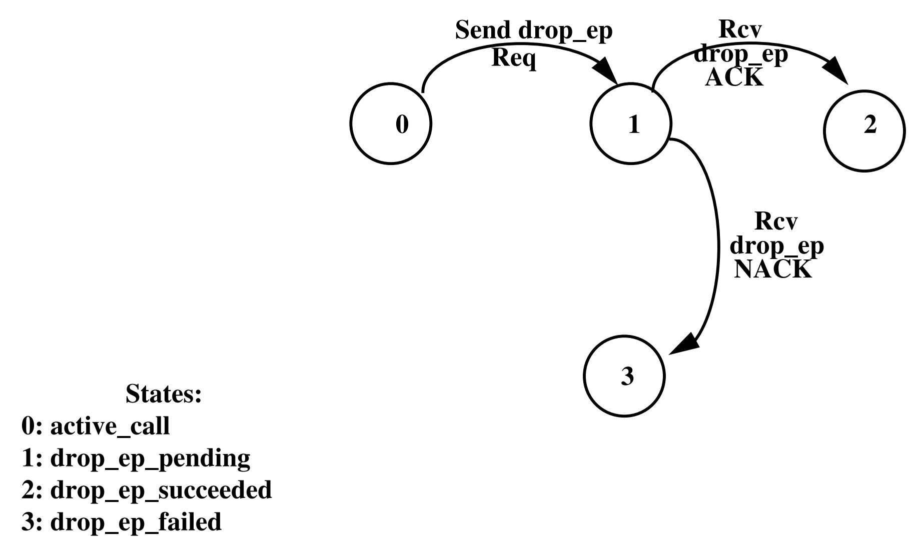
If the owner of a call requests that it be dropped from the call, the operation is treated as if it was a **close_call**, terminating the call.

The owner of a call may request that all endpoints at an specific address be dropped from the call by specifying the ***ep_id*** with all bits set to 1. This would cause multiple **drop_from_call** messages to be sent if multiple endpoints were active at the specified endpoint.

If there are network errors, the **drop_EP** request is acknowledged as successful and it is up to the network to clean up any problems it may have had. A client should not be burdened (i.e. billed) for an endpoint after it has requested to be dropped.

As stated above, the **drop_EP REQUEST** may come from the owner of the call or the endpoint to be dropped. The endpoint making the request, therefore, must be active in the call. The Client CMAP Agent starts in the *active_call* state, generates the **drop_ep REQUEST** and makes the transition to the *drop_ep_pending* state. If the operation is successful, the Client CMAP Agent will receive a positive **drop_ep RESPONSE** and should make the transition to the *drop_ep_succeeded* state. If a negative **drop_ep RESPONSE** is received, the operation was not successful and the ***op_status*** field of the **RESPONSE** will indicate the reason for the failure. The Client CMAP Agent should make the transition to the *drop_ep_failed* state upon receiving a negative **drop_ep RESPONSE**.

Figure 58. Drop Endpoint



7.4.5 drop_from_call

Synopsis:

This operation informs an endpoint that it is being dropped from a call. This is triggered when the owner of the call requests a ***drop_EP*** operation. The REQUEST phase goes from network to client.

Data:

- ***op_type*** - **drop_from_call**
- ***phase*** - defines which type of message this is.

  phase $\in$ {REQUEST}
- ***msg_id*** - message identifier, unique to a client endpoint
- ***call_id*** - call identifier of the call from which the specified endpoint has been dropped. This contains the address of the owner of the call and can be used to determine if the owner was also the requester.
- ***reason -*** reason that the endpoint was dropped.
- ***ep_addr*** - address of endpoint to be dropped from the call
- ***ep_id*** - identifier of the endpoint being dropped.
- ***r_addr*** - address of endpoint requesting the drop
- ***options_size*** - size of the following ***options*** field. If options_size is 0, then there is no ***options*** field present. The ***options_size*** field is always present.
- ***options*** - variable length field for introduction of new options and parameters. Currently there are no options defined.

Message Formats:

drop_from_call REQUEST

<table>
<tr><td colspan="1">00000101</td><td colspan="1">00000000</td><td colspan="2">msg_id (2)</td><td colspan="2">reserved (2)</td><td colspan="2">reserved (2)</td></tr>
<tr><td colspan="8">call_id(o_addr) (24)</td></tr>
<tr><td colspan="2">call_id(lcid) (2)</td><td colspan="2">unused (2)</td><td colspan="4">reserved (4)</td></tr>
<tr><td colspan="8">r_addr (24)</td></tr>
<tr><td colspan="8">ep_addr (24)</td></tr>
<tr><td colspan="2">ep_id (2)</td><td colspan="2">reserved (2)</td><td colspan="4">reserved (4)</td></tr>
<tr><td colspan="4">options_size (4)</td><td colspan="4">options (options_size)</td></tr>
</table>

Operation:

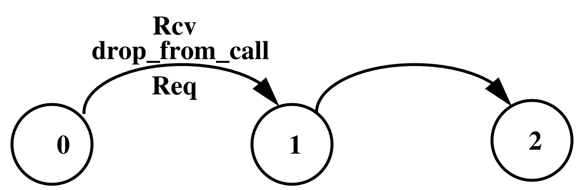
When the owner of a call requests that an endpoint be dropped from the call, a **drop_from_call** operation is triggered. This operation consists of one message sent from the network to the endpoint. Since there is no RESPONSE phase, there is no way for the endpoint to keep from being dropped.

The ***ep_id*** is supplied in case there are multiple endpoints with the same endpoint address, ***ep_addr***.

The ***reason*** field indicates the source and reason for the endpoint being dropped. The possibilities are NETWORK_ERROR, OWNER_REQUEST and CALL_CLOSED.

The Client CMAP Agent (Figure 59) that receives a drop_from_call REQUEST would be in the *active_call* state.

Figure 59. drop from call notification



States:
0: active_call
1: process_drop_from_call
2: no_call

Upon receiving the REQUEST it would make the transition to the *process_drop_from_call* state. At this point the endpoint would no longer be active in the call and should make the transition to the *no_call* state.

7.4.6 invite

Synopsis:

This operation invites an endpoint to join a call and is triggered when a client requests an ***add_ep*** operation. The REQUEST phase goes from network to owner.

Data:

- ***op_type*** - **invite**
- ***phase*** - defines which type of message this is.

  phase ∈ {REQUEST, RESPONSE, CONFIRMATION}
- ***msg_id*** - message identifier, unique to a client endpoint
- ***call_id*** - call identifier of the call to which the specified endpoint is being invited. This contains the address of the owner of the call which may be used to determine if the owner is also the requester.
- ***num_cons*** - specifies the number of connections being specified in the **invite**.
- ***ep_addr*** - address of endpoint being invited to join the call
- ***r_addr*** - address of endpoint requesting this addition. This is needed so that the endpoint being invited can be told the full address, including any local extension.
- ***ep_id*** - endpoint identifier for the endpoint being invited.
- ***op_status*** **-** describes the reason for the failure of the operation.(Section 7.3.23)

  op_status* = <*call_status_bit ∈ {CALL_SPEC_OK, CALL_SPEC_ERROR},
  connection_status_bit ∈ {CON_SPEC_OK, CON_SPEC_ERROR},
  status ∈ {OK, REFUSED,
  BAD_PHASE_EXP_RESPONSE,
  TIMEOUT, ERROR, NEGOTIATING}>
- ***options_size*** - size of the following ***options*** field. If options_size is 0, then there is no ***options*** field present. The ***options_size*** field is always present.
- ***options*** - variable length field for introduction of new options and parameters. Currently there are no options defined.

For each connection specified in the **invite REQ**, a connection specification is given which consists of the following information:

- ***con_id*** - connection identifier, uniquely identifies a connection within a call
- ***con_type*** - specifies what type of connection this is. This field is a 3-tuple.

  con_type = < channel_type ∈ {VP, VC},
  bw_ type ∈ {STATIC, DYNAMIC},
  qos ∈ {HIGH, MEDIUM, LOW}>
- ***con_perm*** - default permission specification for this connection

  con_perm = < receive = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}>,
  transmit = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}> >
- ***ep_perm*** - default endpoint permission specification for this connection.

  ep_perm = < receive = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}>,
  transmit = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}> >
- ***ep_map*** - specifies the UNI mapping to be used for the additional endpoint for this connection.

ep_perm = < receive = < perm ∈ {ON, HOLD, OFF, HOLD}, changeability ∈ {ON, VERIFY, OFF}>,

transmit = < perm ∈ {ON, HOLD, OFF, HOLD}, changeability ∈ {ON, VERIFY, OFF}>,

echo ∈ {ON, OFF} >

- ***notify*** - specifies whether endpoints should be notified if this connection is changed via a **mod_con** operation (Section 7.4.13).

  notify ∈ {**YES**, **NO**}

- ***bw*** - bandwidth specification for the connection.

  bw = <peak, average, peak_burst_length>

- ***user_con_type*** - user description of the type of this connection. Not manipulated by the network, just passed end to end.
- ***trans_vpi*** - transmit virtual path identifier for the new endpoint on this connection
- ***trans_vci*** - transmit virtual channel identifier for the new endpoint on this connection.
- ***rcv_vpi*** - receive virtual path identifier for the new endpoint on this connection
- ***rcv_vci*** - receive virtual channel identifier for the new endpoint on this connection.
- ***con_status*** - this field is used to indicate what was potentially wrong with specific connection specifications.(-Section 7.3.24)

  con_status ∈ {OK,
  DUP_CON_ID, BAD_CON_TYPE, BAD_NOTIFY,
  NO_AVAIL_VCI, RCV_VCI_IN_USE, TRANS_VCI_IN_USE,
  RCV_VCI_RESERVED, RCV_VCI_NOT_SUPPORTED,
  TRANS_VCI_RESERVED, TRANS_VCI_NOT_SUPPORTED,
  NO_AVAIL_VPI, RCV_VPI_IN_USE, TRANS_VPI_IN_USE,
  RCV_VPI_RESERVED, RCV_VPI_NOT_SUPPORTED,
  TRANS_VPI_RESERVED, TRANS_VPI_NOT_SUPPORTED,
  ILL_PERM_CHANGE, BAD_EP_PERM, BAD_BW}

Message Formats:

invite REQUEST

00000110	00000000	msg_id (2)		reserved (2)		reserved (2)	
call_id(o_addr) (24)							
call_id(lcid) (2)		reserved (2)		user_call_type (4)			
unused (1)	unused (1)	mon (1)	unused(1)	unused(1)	reserved (1)	reserved (2)	
num_cons (2)		reserved (2)		ep_id (2)		reserved (2)	
r_addr (24)							
con_id (2)		con_type (1)	notify (1)	bw (12)			
user_con_type (2)		unused (2)		con_perm(1)	ep_perm(1)	ep_map(1)	unused (1)
reserved (1)	trans_vpi(1)	trans_vci (2)		reserved (1)	rcv_vpi (1)	rcv_vci (2)	
⋮							
con_id (2)		con_type (1)	notify (1)	bw (12)			
user_con_type (2)		unused (2)		con_perm(1)	ep_perm(1)	ep_map(1)	unused (1)
reserved (1)	trans_vpi(1)	trans_vci (2)		reserved (1)	rcv_vpi (1)	rcv_vci (2)	
options_size (4)				options (options_size)			

Message Formats (cont):

invite RESPONSE

00000110	00000001	msg_id (2)		reserved (2)		reserved (2)	
call_id(o_addr)							
call_id(lcid) (2)		op_status (2)		ep_id (2)		num_cons (2)	
con_id (2)		con_type (1)	notify (1)	bw (12)			
user_con_type (2)		con_status (2)		con_perm(1)	ep_perm(1)	ep_map(1)	unused (1)
reserved (1)	trans_vpi(1)	trans_vci (2)		reserved (1)	rcv_vpi (1)	rcv_vci (2)	
• • •							
con_id (2)		con_type (1)	notify (1)	bw (12)			
user_con_type (2)		con_status (2)		con_perm(1)	ep_perm(1)	ep_map(1)	unused (1)
reserved (1)	trans_vpi(1)	trans_vci (2)		reserved (1)	rcv_vpi (1)	rcv_vci (2)	
options_size (4)				options (options_size)			

Message Formats (cont):

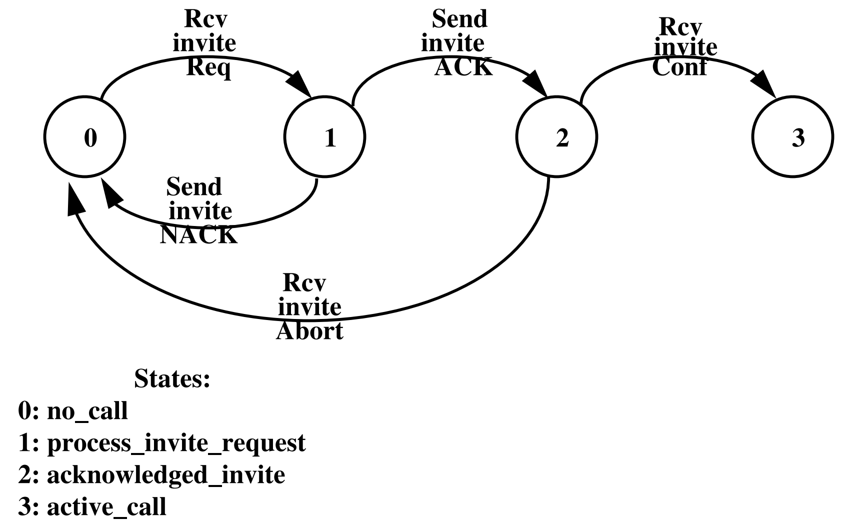
invite CONFIRMATION

00000110	00000010	msg_id (2)		reserved (2)		reserved (2)	
call_id(o_addr)							
call_id(lcid) (2)		op_status (2)		ep_id (2)		num_cons (2)	
con_id (2)		con_type (1)	notify (1)	bw (12)			
user_con_type (2)		con_status (2)		con_perm(1)	ep_perm(1)	ep_map(1)	unused (1)
reserved (1)	trans_vpi(1)	trans_vci (2)		reserved (1)	rcv_vpi (1)	rcv_vci (2)	
• • •							
con_id (2)		con_type (1)	notify (1)	bw (12)			
user_con_type (2)		con_status (2)		con_perm(1)	ep_perm(1)	ep_map(1)	unused (1)
reserved (1)	trans_vpi(1)	trans_vci (2)		reserved (1)	rcv_vpi (1)	rcv_vci (2)	
options_size (4)				options (options_size)			

Operation:

An **invite** operation is triggered when an endpoint issues an **add_ep** REQUEST. The purpose of the **invite** operation is to ask the new endpoint whether it wants to participate in the call..

Figure 60. Endpoint being invited to join a call



Since the endpoint is being invited to join the call the Client CMAP Agent is assumed to be in the *no_call* state when it receives the **invite REQUEST**. Upon receiving the **REQUEST**, the Client CMAP Agent should make the transition to the *process_invite_request* state. The information about each accessible connection in the call is provided in the **REQUEST.** This gives the invited endpoint the information to decide whether it wants to join the call and whether it has the required resources.

A connection is inaccessible if its ep_perm is set as follows:

ep_perm = < receive = < perm = OFF, changeability = OFF>,
transmit = < perm = OFF, changeability = OFF>,
echo $\in$ {ON, OFF} >

If it does not want to join the call, the Client CMAP Agent should generate a negative **invite RESPONSE** (***op_-status*** field of **REFUSED**) and make the transition back to the *no_call* state.

If it does want to join the call, the Client CMAP Agent should generate a positive **invite RESPONSE** (***op_status*** field of **OK**) and make the transition to the *acknowledged_invite* state.

The ep_id specified in the **REQUEST** phase is only a suggestion. The endpoint being invited to join the call may specify a different endpoint identifier in the **invite RESPONSE** (Section 6.4).

If any of the VPI/VCI pairs specified in the REQUEST phase are not acceptable to the endpoint being added, it may change them in the acknowledgment phase(Section 6.4). When the client wishes to make changes to the VPI/VCI pairs, it sets the ***op_status*** field to NEGOTIATING. If the network interface node does not accept these changes, the operation fails and the endpoint receives a negative **invite** CONFIRMATION. The Client CMAP Agent would then make the transition back to the *no_call* state.

The endpoint can also modify its permissions, **ep_perm**, for any of the connections in the call (Section 6.4). Changes to the permissions may trigger a **verify** operation (Section 7.4.7). If the owner does not accept the addition of the endpoint or the changes it requested, the operation fails and the endpoint receives a negative **invite** CONFIRMATION. The Client CMAP Agent would then make the transition back to the *no_call* state.

If it does not want to join the call, the Client CMAP Agent should generate a negative **invite RESPONSE** (***op_-status*** field of **REFUSED**).

7.4.7 verify

Synopsis:

This operation queries the call owner to see if a new endpoint should be added to a call. This is triggered when an endpoint other than the owner of a call requests an ***add_ep*** operation on a call that has the accessibility parameter, ***acc***, set to VERIFY. The REQUEST phase goes from network to owner.

Data:

- ***op_type*** - **verify**
- ***phase*** - defines which type of message this is.

  $$\textbf{\textit{phase}} \in \{\text{REQUEST, RESPONSE}\}$$

- ***msg_id*** - message identifier, unique to a client endpoint
- ***call_id*** - call identifier of the call to which the specified endpoint is being added.
- ***op_status* -** {OK, REFUSED, bad_parm, timeout, ERROR}
- ***num_cons*** - specifies the number of connections being specified in the **verify**.
- ***ep_addr*** - address of the endpoint that is to be added to the call.
- ***r_addr*** - address of endpoint requesting this addition. This is needed so that the owner can be told the full address, including any local extension.
- ***options_size*** - size of the following ***options*** field. If options_size is 0, then there is no ***options*** field present. The ***options_size*** field is always present.
- ***options*** - variable length field for introduction of new options and parameters. Currently there are no options defined.

For each connection specified in the **verify REQ**, a connection specification is given which consists of the following information:

- ***con_id*** - connection identifier, uniquely identifies a connection within a call
- ***ep_perm*** - default endpoint permission specification for this connection.

  $$\begin{aligned} \textbf{\textit{ep\_perm}} = \Big\langle\ & \text{receive} = \langle \text{perm} \in \{\text{ON, HOLD, OFF}\}, \text{changeability} \in \{\text{ON, VERIFY, OFF}\}\rangle, \\ & \text{transmit} = \langle \text{perm} \in \{\text{ON, HOLD, OFF}\}, \text{changeability} \in \{\text{ON, VERIFY, OFF}\}\rangle \Big\rangle \end{aligned}$$

- ***ep_map*** - specifies the UNI mapping to be used for the additional endpoint for this connection.

  $$\begin{aligned} \textbf{\textit{ep\_perm}} = \Big\langle\ & \text{receive} = \langle \text{perm} \in \{\text{ON, HOLD, OFF}\}, \text{changeability} \in \{\text{ON, VERIFY, OFF}\}\rangle, \\ & \text{transmit} = \langle \text{perm} \in \{\text{ON, HOLD, OFF}\}, \text{changeability} \in \{\text{ON, VERIFY, OFF}\}\rangle, \\ & \text{echo} \in \{\text{ON, OFF}\} \Big\rangle \end{aligned}$$

- ***notify*** - specifies whether endpoints should be notified if this connection is changed via a **mod_con** operation (Section 7.4.13).

Message Formats:

verify REQUEST

00000111	00000000	msg_id (2)	reserved (2)	reserved (2)	
call_id(o_addr) (24)					
call_id(lcid) (2)	unused (2)	reserved (2)	num_cons (2)		
r_addr (24)					
ep_addr (24)					
ep_id (2)	reserved (2)	reserved (4)			
con_id (2)	unused (1)	unused (1)	bw (12)		
user_con_type (2)	unused (2)	unused (1)	ep_perm(1)	ep_map(1)	unused (1)
reserved (1)	trans_vpi(1)	trans_vci (2)	reserved (1)	rcv_vpi (1)	rcv_vci (2)
• • •					
con_id (2)	unused (1)	unused (1)	bw (12)		
user_con_type (2)	unused (2)	unused (1)	ep_perm(1)	ep_map(1)	unused (1)
reserved (1)	trans_vpi(1)	trans_vci (2)	reserved (1)	rcv_vpi (1)	rcv_vci (2)
options_size (4)	options (options_size)				

Message Formats (cont):

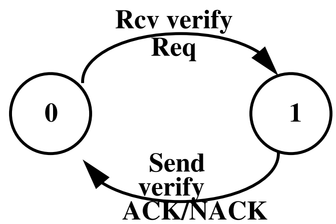
verify RESPONSE

00000111	00000001	msg_id (2)		reserved (2)		reserved (2)	
call_id(o_addr) (24)							
call_id(lcid) (2)		unused (2)		reserved (2)		num_cons (2)	
ep_addr (24)							
ep_id (2)		reserved (2)		reserved (4)			
con_id (2)		unused (1)	unused (1)	bw (12)			
user_con_type (2)		con_status (2)		unused (1)	ep_perm(1)	ep_map(1)	unused (1)
reserved (1)	trans_vpi(1)	trans_vci (2)		reserved (1)	rcv_vpi (1)	rcv_vci (2)	
• • •							
con_id (2)		unused (1)	unused (1)	bw (12)			
user_con_type (2)		con_status (2)		unused (1)	ep_perm(1)	ep_map(1)	unused (1)
reserved (1)	trans_vpi(1)	trans_vci (2)		reserved (1)	rcv_vpi (1)	rcv_vci (2)	
options_size (4)				options (options_size)			

Operation:

When an **add_ep** operation is requested by an endpoint other than the owner of the call, and the call has the accessibility parameter set to **VERIFY**, a **verify** operation is triggered. Also, if the endpoint being added is not the owner and the endpoint permissions for one or more connections resulting from the **add_ep** and **invite** operations are other than the default and the changeability subfield of that connection's permissions is set to **VERIFY**, then a **verify** operation is triggered. A verify operation will not be triggered if the owner generated the add_ep and specified the connection permissions that were different from the default.

Figure 61. Owner verifying the addition of a new endpoint



States:
0: active_call
1: process_verify_request

Since the owner of the call is already active in the call, the Client CMAP Agent is in the *active_call* state when it receives the **verify REQUEST**. Upon receiving the **REQUEST**, the Client CMAP Agent makes the transition to the *process_verify_request* state.

If the owner wishes to allow the endpoint to join the call, the Client CMAP Agent generates a positive **verify RESPONSE** (***op_status*** of **OK**) and makes the transition back to the *active_call* state.

If the owner does not wish to allow the endpoint to join the call, the Client CMAP Agent generates a negative **verify RESPONSE** (***op_status*** of **REFUESED**) and makes the transition back to the *active_call* state.

7.4.8 announce_EP

Synopsis:

This operation informs a call owner that a new endpoint has joined the call. The REQUEST phase goes from network to owner.

Data:

- ***op_type*** - **announce_EP**
- ***phase*** -defines which type of message this is.

  phase ∈ {REQUEST}
- ***msg_id*** - message identifier, unique to a client endpoint
- ***call_id*** - call identifier of the call to which the specified endpoint has been added.
- ***ep_addr*** - address of new endpoint.
- ***r_addr*** - address of client requesting the **add_ep**.
- ***ep_id*** - endpoint identifier for the endpoint joining the call.
- ***options_size*** - size of the following ***options*** field. If options_size is 0, then there is no ***options*** field present. The ***options_size*** field is always present.
- ***options*** - variable length field for introduction of new options and parameters. Currently there are no options defined.

For each connection specified in the **announce_EP REQUEST**, a connection specification is given which consists of the following information:

- ***con_id*** - connection identifier, uniquely identifies a connection within a call
- ***con_type*** - specifies what type of connection this is. This field is a 3-tuple.

  con_type = < channel_type ∈ {VP, VC},
  bw_ type ∈ {STATIC, DYNAMIC},
  qos ∈ {HIGH, MEDIUM, LOW}>
- ***con_perm*** - default permission specification for this connection

  con_perm = < receive = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}>,
  transmit = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}> >
- ***ep_perm*** - default endpoint permission specification for this connection.

  ep_perm = < receive = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}>,
  transmit = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}> >
- ***ep_map*** - specifies the UNI mapping to be used for the additional endpoint for this connection.

  ep_perm = < receive = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}>,
  transmit = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}>,
  echo ∈ {ON, OFF} >
- ***notify*** - specifies whether endpoints should be notified if this connection is changed via a **mod_con** operation (Section 7.4.13).

  notify ∈ {**YES**, **NO**}
- ***bw*** - bandwidth specification for the connection.

  bw = <peak, average, peak_burst_length>

- ***user_con_type*** - user description of the type of this connection. Not manipulated by the network, just passed end to end.
- ***trans_vpi*** - transmit virtual path identifier for the owner on this connection
- ***trans_vci*** - transmit virtual channel identifier for the owner on this connection.
- ***rcv_vpi*** - receive virtual path identifier for the owner on this connection
- ***rcv_vci*** - receive virtual channel identifier for the owner on this connection.

Message Formats:

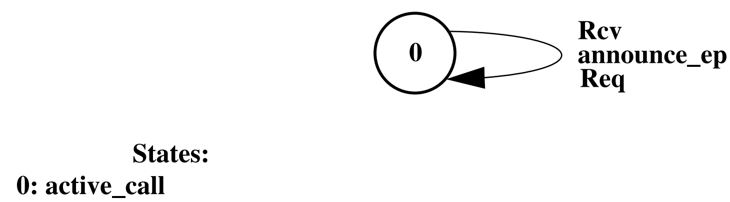
announce_ep REQUEST

00001000	00000000	msg_id (2)		reserved (2)		reserved (2)	
call_id(o_addr) (24)							
call_id(lcid)		unused (2)		reserved (2)		num_cons (2)	
r_addr (24)							
ep_addr (24)							
ep_id (2)		reserved (2)		reserved (4)			
con_id (2)		con_type (1)	notify (1)	bw (12)			
user_con_type (2)		unused (2)		con_perm(1)	ep_perm(1)	ep_map(1)	unused (1)
reserved (1)	trans_vpi(1)	trans_vci (2)		reserved (1)	rcv_vpi (1)	rcv_vci (2)	
• • •							
con_id (2)		con_type (1)	notify (1)	bw (12)			
user_con_type (2)		unused (2)		con_perm(1)	ep_perm(1)	ep_map(1)	unused (1)
reserved (1)	trans_vpi(1)	trans_vci (2)		reserved (1)	rcv_vpi (1)	rcv_vci (2)	
options_size (4)				options (options_size)			

Operation:

An **announce_EP REQUEST** is triggered by an endpoint other than the owner performing an ***add_ep*** operation on a call that has the monitor endpoint parameter, *mon*, set to YES. There are no state transitions for the Client CMAP Agent of the owner in response to an ***announce_EP*** **REQUEST**.

Figure 62. Owner being notified of a new endpoint in the call



The owner is informed of the endpoint permissions assigned to the new endpoint for each connection in the call.

7.4.9 announce_drop

Synopsis:

This operation informs the owner of a call that an endpoint has dropped out. The REQUEST phase goes from network to owner.

Data:

- ***op_type*** - **announce_drop**
- ***phase*** -defines which type of message this is.

  phase $\in$ {REQUEST}
- ***msg_id*** - message identifier, unique to a client endpoint
- ***call_id*** - call identifier of the call from which the specified endpoint was dropped.
- ***ep_addr*** - address of endpoint that dropped out.
- ***ep_id*** - endpoint identifier for the endpoint that dropped out of the call.
- ***reason* -** reason that the endpoint was dropped.

  reason $\in$ {NETWORK_ERROR, ENDPOINT_REQUEST}
- ***options_size*** - size of the following ***options*** field. If options_size is 0, then there is no ***options*** field present. The ***options_size*** field is always present.
- ***options*** - variable length field for introduction of new options and parameters. Currently there are no options defined.

Message Formats:

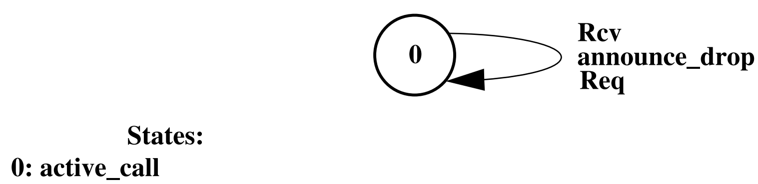
announce_drop REQUEST

<table>
<tr><td>00001001</td><td>00000000</td><td colspan="2">msg_id (2)</td><td colspan="2">reserved (2)</td><td colspan="2">reserved (2)</td></tr>
<tr><td colspan="8">call_id(o_addr) (24)</td></tr>
<tr><td colspan="2">call_id(lcid) (2)</td><td colspan="2">unused (2)</td><td>reason (1)</td><td>reserved (1)</td><td colspan="2">reserved (2)</td></tr>
<tr><td colspan="8">ep_addr (24)</td></tr>
<tr><td colspan="2">ep_id (2)</td><td colspan="2">reserved (2)</td><td colspan="4">reserved (4)</td></tr>
<tr><td colspan="4">options_size (4)</td><td colspan="4">options (options_size)</td></tr>
</table>

Operation:

The announce_drop message is normally triggered by an endpoint other than the owner performing a ***drop_EP*** operation for a call that has the monitor endpoint parameter, *mon*, set to YES.

Figure 63. Owner being notified of an endpoint dropping out of the call



The **announce_drop** message may also be triggered by an error in the network causing an endpoint to be dropped.

There are no state transitions for the Client CMAP Agent of the owner, in response to an ***announce_drop*** **REQUEST**

7.4.10open_con

Synopsis:

This operation requests that a new connection be opened within an existing call. This can only be requested by the owner of the call. The REQUEST phase goes from owner to network.

Data:

- ***op_type*** - **open_con**
- ***phase*** - defines which type of message this is.

  phase ∈ {REQUEST, RESPONSE}
- ***msg_id*** - message identifier, unique to a client endpoint
- ***call_id*** - call identifier of the call to which the connection is to be added.
- ***num_cons*** - specifies the number of connections being specified in the **open_con**.
- ***op_status*** - describes the reason for the failure of the operation.(Section 7.3.23)

  op_status = <call_status_bit ∈ {CALL_SPEC_OK, CALL_SPEC_ERROR***},***
  connection_status_bit ∈ {CON_SPEC_OK, CON_SPEC_ERROR***},***
  status ∈ {OK,
  INSUFF_BW, NOT_OWNER,
  DUP_MSG_ID, BAD_MSG_ID,
  BAD_PHASE_EXP_REQUEST,
  TIMEOUT}>
- ***options_size*** - size of the following ***options*** field. If options_size is 0, then there is no ***options*** field present. The ***options_size*** field is always present.
- ***options*** - variable length field for introduction of new options and parameters. Currently there are no options defined.

For the connection specified in the **open_con REQ**, a connection specification is given which consists of the following information:

- ***con_id*** - connection identifier, uniquely identifies the new connection within a call
- ***con_type*** - specifies what type of connection this is. This field is a 3-tuple.

  con_type = < channel_type ∈ {VP, VC},
  bw_ type ∈ {STATIC, DYNAMIC},
  qos ∈ {HIGH, MEDIUM, LOW}>
- ***con_perm*** - default permission specification for this connection

  con_perm = < receive = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}>,
  transmit = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}> >
- ***crtr_map*** - specifies the UNI mapping to be used for the creator of this connection.

  crtr_map = < receive = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}>,
  transmit = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}>,
  echo ∈ {ON, OFF} >
- ***notify*** - specifies whether endpoints should be notified if this connection is changed via a **mod_con** operation (Section 7.4.13).

  notify ∈ {**YES**, **NO**}

- ***bw*** - bandwidth specification for the connection.

  bw = <peak, average, peak_burst_length>

- ***user_con_type*** - user description of the type of this connection. Not manipulated by the network, just passed end to end.
- ***trans_vpi*** - transmit virtual path identifier for the owner on this connection
- ***trans_vci*** - transmit virtual channel identifier for the owner on this connection.
- ***rcv_vpi*** - receive virtual path identifier for the owner on this connection
- ***rcv_vci*** - receive virtual channel identifier for the owner on this connection.
- ***con_status*** - this field is used to indicate what was potentially wrong with specific connection specifications. (Section 7.3.24)

  con_status ∈ {OK,
  DUP_CON_ID, BAD_CON_TYPE, BAD_NOTIFY,
  NO_AVAIL_VCI, RCV_VCI_IN_USE, TRANS_VCI_IN_USE,
  RCV_VCI_RESERVED, RCV_VCI_NOT_SUPPORTED,
  TRANS_VCI_RESERVED, TRANS_VCI_NOT_SUPPORTED,
  NO_AVAIL_VPI, RCV_VPI_IN_USE, TRANS_VPI_IN_USE,
  RCV_VPI_RESERVED, RCV_VPI_NOT_SUPPORTED,
  TRANS_VPI_RESERVED, TRANS_VPI_NOT_SUPPORTED,
  ILL_PERM_CHANGE,
  BAD_CON_PERM, BAD_OWN_PERM, BAD_BW}

Message Formats:

open_con REQUEST

00001010	00000000	msg_id (2)		reserved (2)		reserved (2)	
call_id(o_addr) (24)							
call_id(lcid) (2)		unused (2)		reserved (2)		num_cons (2)	
r_addr (24)							
con_id (2)		con_type (1)	notify (1)	bw (12)			
user_con_type (2)		unused (2)		con_perm(1)	ep_perm(1)	ep_map(1)	ctrt_map (1
reserved (1)	trans_vpi(1)	trans_vci (2)		reserved (1)	rcv_vpi (1)	rcv_vci (2)	
• • •							
con_id (2)		con_type (1)	notify (1)	bw (12)			
user_con_type (2)		unused (2)		con_perm(1)	ep_perm(1)	ep_map(1)	ctrt_map (1
reserved (1)	trans_vpi(1)	trans_vci (2)		reserved (1)	rcv_vpi (1)	rcv_vci (2)	
options_size (4)				options (options_size)			

Message Formats (cont):

open_con RESPONSE

00001010	00000001	msg_id (2)		reserved (2)		reserved (2)	
call_id(o_addr) (24)							
call_id(lcid) (2)		op_status (2)		reserved (2)		num_cons (2)	
con_id (2)		con_type (1)	notify (1)	bw (12)			
user_con_type (2)		con_status (2)		con_perm(1)	ep_perm(1)	ep_map(1)	ctrt_map (1
reserved (1)	trans_vpi(1)	trans_vci (2)		reserved (1)	rcv_vpi (1)	rcv_vci (2)	
• • •							
con_id (2)		con_type (1)	notify (1)	bw (12)			
user_con_type (2)		con_status (2)		con_perm(1)	ep_perm(1)	ep_map(1)	ctrt_map (1
reserved (1)	trans_vpi(1)	trans_vci (2)		reserved (1)	rcv_vpi (1)	rcv_vci (2)	
options_size (4)				options (options_size)			

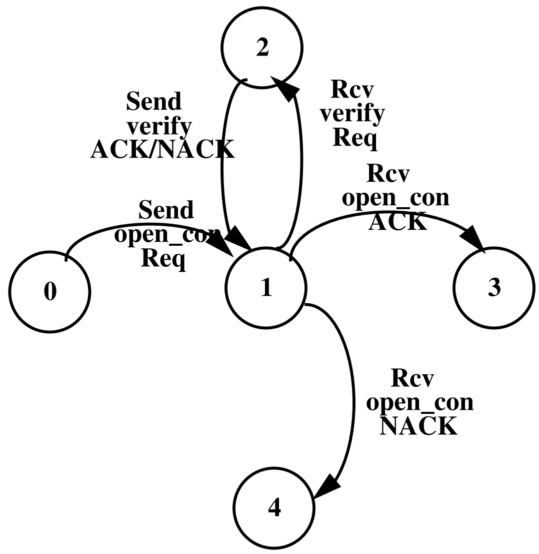
Operation:

The owner of a call requests that one or more new connections be added to the call using the **open_con** operation.

The owner may specify the VPI/VCI pairs for the its access link to be used for the new connections. Different VPI/VCI pairs may be used for the receive and transmit portions of each connection. If VPI/VCI pair selected by the client is not acceptable to the network the **open_con** fails and a negative acknowledgment is returned to the client.

Since the owner is already active in the call, the Client CMAP Agent starts in the **active_call** state, generates an **open_con REQUEST** and makes the transition to the **processing_open_con** state.

Figure 64. Owner adding new connections to a call



States:
0: active_call
1: processing_open_con
2: open_con_verifying
2: open_con_successful
3: open_con_failed

In the process of adding the new connections, all the endpoints active in the call will receive an **open_con_invite REQUEST**, inviting them to participate in the new connections. If any of the permissions on the new connections have the changeability subfield set to VERIFY, then for each endpoint that requests to change the permissions offered to it for any of those connection, the owner will receive a **verify REQUEST** (Section 7.4.7).

If the connections can be set up, the network acknowledges, specifying the VPI/'VCI pairs selected in a positive **open_con RESPONSE**.

A negative **open_con RESPONSE** contains information in the ***op_status*** and ***con_status*** fields that indicates the reason for the failure to set up the new connections.

NB: An extension to the open_con operation that is under consideration at this time is to allow non-owner endpoints to request the operation. This would be useful in a multi-transmitter video conference. As each new transmitter adds itself to the call, it also adds a connection for its transmission.

7.4.11close_con

Synopsis:

This operation requests that an existing connection within a call be closed. This can only be requested by the owner of the call. The REQUEST phase goes from owner to network.

Data:

- ***op_type*** - **close_con**
- ***phase*** - defines which type of message this is.

  phase ∈ {REQUEST, RESPONSE}
- ***msg_id*** - message identifier, unique to a client endpoint
- ***op_status* -** describes the reason for the failure of the operation.(Section 7.3.23)

  op_status = <call_status_bit ∈ {CALL_SPEC_OK, CALL_SPEC_ERROR},

  connection_status_bit ∈ {CON_SPEC_OK, CON_SPEC_ERROR},

  status ∈ {OK,

  NOT_OWNER,

  DUP_MSG_ID, BAD_MSG_ID,

  BAD_PHASE_EXP_REQUEST,

  TIMEOUT}>
- ***call_id*** - call identifier of the call.
- ***con_id*** - the connection identifier of the connection to be closed.
- ***options_size*** - size of the following ***options*** field. If options_size is 0, then there is no ***options*** field present. The ***options_size*** field is always present.
- ***options*** - variable length field for introduction of new options and parameters. Currently there are no options defined.

Message Formats:

close_con REQUEST

00001011	00000000	msg_id (2)	reserved (2)	reserved (2)
call_id(o_addr) (24)				
call_id(lcid) (2)		unused (2)	con_id (2)	reserved (2)
options_size (4)			options (options_size)	

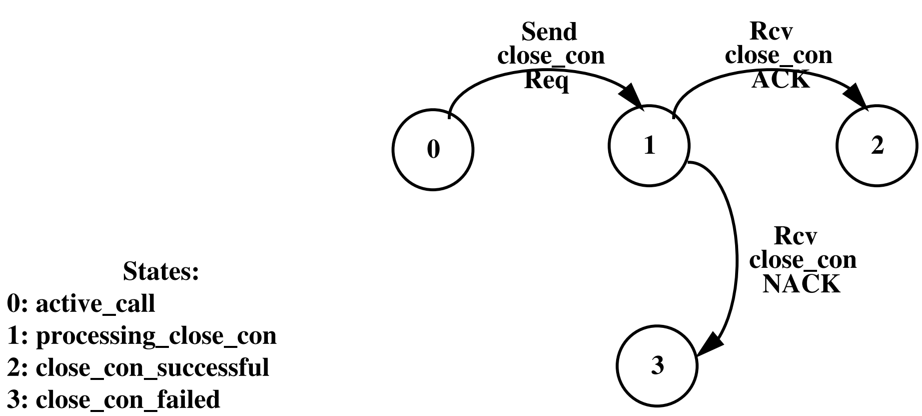
close_con RESPONSE

00001011	00000001	msg_id (2)	reserved (2)	reserved (2)
call_id(o_addr) (24)				
call_id(lcid) (2)		op_status (2)	con_id (2)	reserved (2)
options_size (4)			options (options_size)	

Operation:

The owner initiates a **close_con** operation to remove a connection from a call. Since the owner is necessarily already active in the call the Client CMAP Agent starts in the ***active_call*** state, generates the **close_con REQUEST** and makes the transition to the ***processing_close_con*** state. The **REQUEST** specifies the connection identifier (***con_id***)

Figure 65. Owner closing a connection within a call



of the connection to be removed. If the connection is removed successfully the network will send a positive **open_con RESPONSE** to the owner of the call which will then make the transition to the ***close_con_successful*** state. All the endpoints in the call will receive an **announce_change** message before the connection is actually dropped so they can stop transmitting.

If any errors occur, the connection will not be removed, the owner will receive a negative open_con RESPONSE and will make the transition to the ***close_con_failed*** state. The ***op_status*** field will give some indication as to the type of problem. The endpoints will receive no notification.

If the connection to be closed is the last one in the call, the operation is treated as if it was a **close_call**, terminating the call.

7.4.12mod_call

Synopsis:

This operation requests a change in the general call parameters. The REQUEST phase goes from owner to network.

Data:

- ***op_type*** - **mod_call**
- ***phase*** - defines which type of message this is.

  phase ∈ {REQUEST, RESPONSE}
- ***msg_id*** - message identifier, unique to a client endpoint
- ***call_id*** - call identifier of call to be modified
- ***user_call_type*** - user description of the type of this call. Not manipulated by the network, just passed end to end.
- ***call_type*** - specifies what type of call this is.

  call_type ∈ {POINT_TO_POINT, MULTIPOINT}
- ***acc*** - specifies whether endpoints may freely join the call.

  acc ∈ {OPEN, VERIFY, CLOSED}
- ***mon*** - specifies who should be notified when endpoints join, drop or are modified in the call.

  mon = <owner ∈ {ON, OFF}, transmitters ∈ {ON, OFF}, all ∈ {ON, OFF} >
- ***modifiability*** - specifies whether endpoints other than the owner may add connections to this call.

  modifiability ∈ {ON, OFF}
- ***priority*** - specifies a priority level for this call.

  priority ∈ {NORMAL, PREEMPT, OVERRIDE}
- ***op_status* -** describes the reason for the failure of the operation. (Section 7.3.23)

  op_status* = <*call_status_bit ∈ {CALL_SPEC_OK, CALL_SPEC_ERROR},
  connection_status_bit ∈ {CON_SPEC_OK, CON_SPEC_ERROR},
  status ∈ {OK, BAD_CALL_ID_ADDR,
  DUP_MSG_ID, BAD_MSG_ID,
  BAD_PHASE_EXP_REQUEST,
  BAD_ACC, BAD_MON,
  BAD_PRIORITY, BAD_MODIFIABLE,
  TIMEOUT}>
- ***options_size*** - size of the following ***options*** field. If options_size is 0, then there is no ***options*** field present. The ***options_size*** field is always present.
- ***options*** - variable length field for introduction of new options and parameters. Currently there are no options defined.

Message Formats:

mod_call REQUEST

00001100	00000000	msg_id (2)		reserved (2)		reserved (2)	
call_id(o_addr) (24)							
call_id(lcid) (2)		unused (2)		user_call_type (4)			
call_type (1)	acc (1)	mon (1)	modifiable(1)	priority(1)	reserved (1)	reserved (2)	
options_size (4)				options (options_size)			

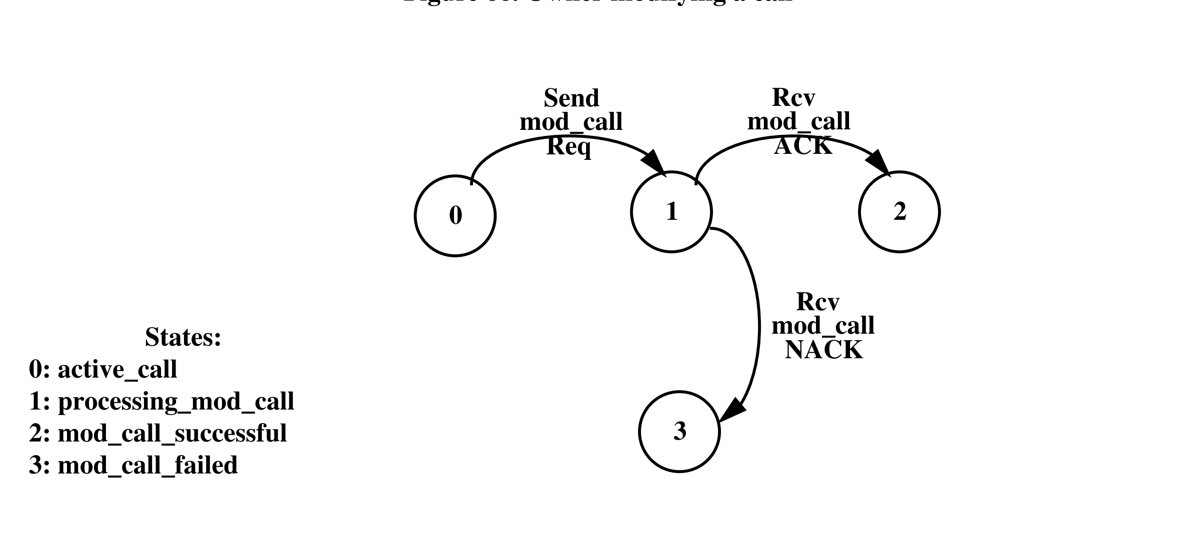
mod_call RESPONSE

00001100	00000001	msg_id (2)	reserved (2)	reserved (2)
call_id(o_addr) (24)				
call_id(lcid) (2)		op_status (2)	reserved (4)	
options_size (4)			options (options_size)	

Operation:

The owner of a call changes the parameters of a call by issuing a **mod_call REQUEST**. The parameters that may be changed are: acc, monitor, priority, user_call_type and modifiability.

Figure 66. Owner modifying a call



Since the owner is necessarily already active in the call the Client CMAP Agent starts in the ***active_call*** state, generates the **mod_call REQUEST** and makes the transition to the ***processing_mod_call*** state.

If the modification was successful, the network will send a positive **mod_call RESPONSE** to the owner of the call which will then make the transition to the ***mod_call_successful*** state. If the user_call_type parameter was change, all the endpoints taking part in the call receive notification of that change only through the ***announce_change*** operation.

If the modification was not successful, the network will send a negative **mod_call RESPONSE** to the owner. The ***op_status*** field will give an indication of the reason for the failure. Upon receipt of the negative **RESPONSE** the owner will make the transition to the ***mod_call_failed*** state.

7.4.13mod_con

Synopsis:

This operation modifies an existing connection in a call. The REQUEST phase goes from owner to network.

Data:

- ***op_type*** - **mod_con**
- ***phase*** - defines which type of message this is.

  phase ∈ {REQUEST, RESPONSE}
- ***msg_id*** - message identifier, unique to a client endpoint
- ***call_id*** - call identifier of the call containing the connection being modified.
- ***num_cons*** - specifies the number of connections being specified in the **mod_con**.
- ***op_status -*** describes the reason for the failure of the operation.(Section 7.3.23)

  op_status = <call_status_bit ∈ {CALL_SPEC_OK, CALL_SPEC_ERROR***},***

  connection_status_bit ∈ {CON_SPEC_OK, CON_SPEC_ERROR***},***

  status ∈ {OK,

  INSUFF_BW, NOT_OWNER,

  DUP_MSG_ID, BAD_MSG_ID,

  BAD_PHASE_EXP_REQUEST,

  TIMEOUT}>
- ***options_size*** - size of the following ***options*** field. If options_size is 0, then there is no ***options*** field present. The ***options_size*** field is always present.
- ***options*** - variable length field for introduction of new options and parameters. Currently there are no options defined.

For the connection specified in the **open_con REQ**, a connection specification is given which consists of the following information:

- ***con_id*** - connection identifier, uniquely identifies a connection within a call
- ***con_type*** - specifies what type of connection this is. This field is a 3-tuple.

  con_type = < channel_type ∈ {VP, VC},

  bw_ type ∈ {STATIC, DYNAMIC},

  qos ∈ {HIGH, MEDIUM, LOW}>
- ***con_perm*** - default permission specification for this connection

  con_perm = < receive = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}>,

  transmit = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}> >
- ***notify*** - specifies whether endpoints should be notified if this connection is changed via a **mod_con** operation (Section 7.4.13).

  notify ∈ {**YES**, **NO**}
- ***bw*** - bandwidth specification for the connection.

  bw = <peak, average, peak_burst_length>
- ***user_con_type*** - user description of the type of this connection. Not manipulated by the network, just passed end to end.
- ***con_status*** - this field is used to indicate what was potentially wrong with specific connection specifications. (Section 7.3.24)

con_status ∈ {OK,
DUP_CON_ID, BAD_CON_TYPE, BAD_NOTIFY,
NO_AVAIL_VCI, RCV_VCI_IN_USE, TRANS_VCI_IN_USE,
RCV_VCI_RESERVED, RCV_VCI_NOT_SUPPORTED,
TRANS_VCI_RESERVED, TRANS_VCI_NOT_SUPPORTED,
NO_AVAIL_VPI, RCV_VPI_IN_USE, TRANS_VPI_IN_USE,
RCV_VPI_RESERVED, RCV_VPI_NOT_SUPPORTED,
TRANS_VPI_RESERVED, TRANS_VPI_NOT_SUPPORTED,
ILL_PERM_CHANGE,
BAD_CON_PERM, BAD_EP_PERM,
BAD_OWN_PERM, BAD_BW}

Message Formats:

mod_con REQUEST

00001101	00000000	msg_id (2)		reserved (2)		reserved (2)	
call_id(o_addr) (24)							
call_id(lcid) (2)		unused (2)		reserved (2)		num_cons (2)	
con_id (2)		con_type (1)	notify (1)	bw (12)			
user_con_type (2)		unused (2)		con_perm(1)	unused(1)	unused(1)	unused (1)
reserved (1)	unused(1)	unused (2)		reserved (1)	unused (1)	unused (2)	
⋮							
con_id (2)		con_type (1)	notify (1)	bw (12)			
user_con_type (2)		unused (2)		con_perm(1)	unused(1)	unused(1)	unused (1)
reserved (1)	unused(1)	unused (2)		reserved (1)	unused (1)	unused (2)	
options_size (4)				options (options_size)			

Message Formats (cont):

mod_con RESPONSE

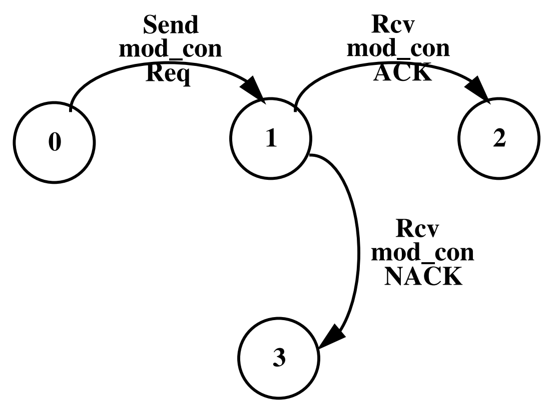
00001101	00000001	msg_id (2)		reserved (2)		reserved (2)	
call_id(o_addr) (24)							
call_id(lcid) (2)		op_status (2)		reserved (2)		num_cons (2)	
con_id (2)		con_type (1)	notify (1)	bw (12)			
user_con_type (2)		con_status (2)		con_perm(1)	unused(1)	unused(1)	unused (1)
reserved (1)	unused(1)	unused (2)		reserved (1)	unused (1)	unused (2)	
• • •							
con_id (2)		con_type (1)	notify (1)	bw (12)			
user_con_type (2)		con_status (2)		con_perm(1)	unused(1)	unused(1)	unused (1)
reserved (1)	unused(1)	unused (2)		reserved (1)	unused (1)	unused (2)	
options_size (4)				options (options_size)			

Operation:

The owner of a call changes the parameters of a connection within that call by issuing a **mod_con REQUEST**. The parameters that may be changed are: con_type, notify, con_perm, user_con_type and bw.

Since the owner is necessarily already active in the call the Client CMAP Agent starts in the ***active_call*** state, generates the **mod_con REQUEST** and makes the transition to the ***processing_mod_con*** state.

Figure 67. Owner modifying connections within a call



States:
0: active_call
1: processing_mod_con
2: mod_con_successful
3: mod_con_failed

If the modification was successful, the network will send a positive **mod_con RESPONSE** to the owner of the call which will then make the transition to the ***mod_con_successful*** state. All the endpoints taking part in the call receive notification of the change through the ***announce_change*** operation if the ***notify*** parameter for any of the modified connections was previously set to **YES**.

If the modification was not successful, the network will send a negative **mod_con RESPONSE** to the owner. The ***op_status*** and ***con_status*** fields will give an indication of the reason for the failure. Upon receipt of the negative **RESPONSE** the owner will make the transition to the ***mod_con_failed*** state.

7.4.14mod_ep

Synopsis:

This operation requests a change in the permissions for an existing endpoint. The REQUEST phase goes from owner to network.

Data:

- ***op_type*** - **mod_ep**
- ***phase*** - defines which type of message this is.

  phase ∈ {REQUEST, RESPONSE}
- ***msg_id*** - message identifier, unique to a client endpoint
- ***status* -** {OK, bad_parm, timeout, ERROR}
- ***call_id*** - call identifier of the call in which the endpoint is to be modified.
- ***ep_addr*** - address of endpoint being modified.
- ***ep_id*** - endpoint identifier for the endpoint being modified.
- ***num_cons*** - specifies the number of connections being specified in the **mod_ep**.
- ***op_status* -** describes the reason for the failure of the operation.(Section 7.3.23)

  op_status = <call_status_bit ∈ ***{***CALL_SPEC_***OK,*** CALL_SPEC_***ERROR},***
  connection_status_bit ∈ ***{***CON_SPEC_***OK,*** CON_SPEC_***ERROR},***
  status ∈ {OK, UNKNOWN_CALL, BAD_CALL_ID_ADDR,
  INSUFF_BW, VERIFY_REFUSED,
  EP_REFUSED, DUP_MSG_ID, BAD_MSG_ID,
  BAD_PHASE_EXP_REQUEST,
  BAD_NUM_CONS, BAD_EP_ADDR, TIMEOUT}>
- ***options_size*** - size of the following ***options*** field. If options_size is 0, then there is no ***options*** field present. The ***options_size*** field is always present.
- ***options*** - variable length field for introduction of new options and parameters. Currently there are no options defined.

For each connection specified in the **mod_ep REQ**, a connection specification is given which consists of the following information:

- ***con_id*** - connection identifier, uniquely identifies a connection within a call
- ***con_type*** - specifies what type of connection this is. This field is a 3-tuple.

  con_type = < channel_type ∈ {VP, VC},
  bw_ type ∈ {STATIC, DYNAMIC},
  qos ∈ {HIGH, MEDIUM, LOW}>
- ***ep_perm*** - default endpoint permission specification for this connection.

  ep_perm = < receive = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}>,
  transmit = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}> >
- ***ep_map*** - specifies the UNI mapping to be used for the additional endpoint for this connection.

  ep_perm = < receive = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}>,
  transmit = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}>,

echo ∈ {ON, OFF} >

- ***notify*** - specifies whether endpoints should be notified if this connection is changed via a **mod_con** operation (Section 7.4.13).

  notify ∈ {**YES**, **NO**}

- ***bw*** - bandwidth specification for the connection.

  bw = <peak, average, peak_burst_length>

- ***user_con_type*** - user description of the type of this connection. Not manipulated by the network, just passed end to end.
- ***trans_vpi*** - transmit virtual path identifier for the endpoint on this connection
- ***trans_vci*** - transmit virtual channel identifier for the endpoint on this connection.
- ***rcv_vpi*** - receive virtual path identifier for the endpoint on this connection
- ***rcv_vci*** - receive virtual channel identifier for the endpoint on this connection.
- ***con_status*** - this field is used to indicate what was potentially wrong with specific connection specifications.(-Section 7.3.24)

  con_status ∈ {OK,
  DUP_CON_ID, BAD_CON_TYPE, BAD_NOTIFY,
  NO_AVAIL_VCI, RCV_VCI_IN_USE, TRANS_VCI_IN_USE,
  RCV_VCI_RESERVED, RCV_VCI_NOT_SUPPORTED,
  TRANS_VCI_RESERVED, TRANS_VCI_NOT_SUPPORTED,
  NO_AVAIL_VPI, RCV_VPI_IN_USE, TRANS_VPI_IN_USE,
  RCV_VPI_RESERVED, RCV_VPI_NOT_SUPPORTED,
  TRANS_VPI_RESERVED, TRANS_VPI_NOT_SUPPORTED,
  ILL_PERM_CHANGE, BAD_EP_PERM, BAD_BW}

Message Formats:

mod_ep REQUEST

00001110	00000000	msg_id (2)		reserved (2)		reserved (2)	
call_id(o_addr) (24)							
call_id(lcid) (2)		unused (2)		reserved (2)		num_cons (2)	
r_addr (24)							
ep_addr (24)							
ep_id (2)		reserved (2)		reserved (4)			
con_id (2)		con_type (1)	notify (1)	bw (12)			
user_con_type (2)		unused (2)		con_perm(1)	ep_perm (1)	ep_map (1)	unused (1)
reserved (1)	unused(1)	unused (2)		reserved (1)	unused (1)	unused (2)	
• • •							
con_id (2)		con_type (1)	notify (1)	bw (12)			
user_con_type (2)		unused (2)		con_perm(1)	ep_perm (1)	ep_map (1)	unused (1)
reserved (1)	unused(1)	unused (2)		reserved (1)	unused (1)	unused (2)	
options_size (4)				options (options_size)			

Message Formats (cont):

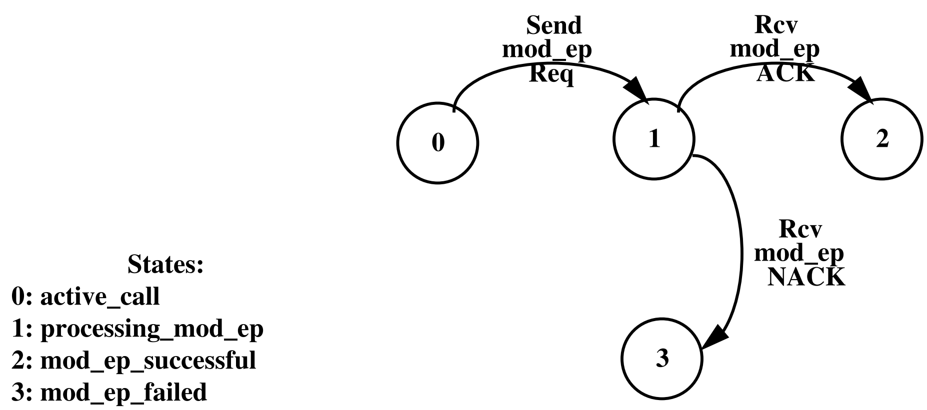
mod_ep RESPONSE

00001110	00000001	msg_id (2)		reserved (2)		reserved (2)	
call_id(o_addr) (24)							
call_id(lcid) (2)		op_status (2)		reserved (2)		num_cons (2)	
ep_addr (24)							
ep_id (2)		reserved (2)		reserved (4)			
con_id (2)		con_type (1)	notify (1)	bw (12)			
user_con_type (2)		con_status (2)		con_perm(1)	ep_perm (1)	ep_map (1)	unused (1)
reserved (1)	unused(1)	unused (2)		reserved (1)	unused (1)	unused (2)	
• • •							
con_id (2)		con_type (1)	notify (1)	bw (12)			
user_con_type (2)		con_status (2)		con_perm(1)	ep_perm (1)	ep_map (1)	unused (1)
reserved (1)	unused(1)	unused (2)		reserved (1)	unused (1)	unused (2)	
options_size (4)				options_size (options_size)			

Operation:

The owner of a call changes the parameters of an endpoints participation in one or more connections within a call by issuing a **mod_ep REQUEST**. The parameters that may be changed are: ep_perm and notify.

Figure 68. Owner modifying an endpoint's permissions



Since the owner is necessarily already active in the call the Client CMAP Agent starts in the ***active_call*** state, generates the **mod_ep REQUEST** and makes the transition to the ***processing_mod_ep*** state. The network will generate a **mod_ep_invite REQUEST** (Section 7.4.14) and sends it to the endpoint being modified. The endpoint then has the opportunity to negotiate the changes and potential new VPI/VCI pairs

If the modification was successful, the network will send a positive **mod_ep RESPONSE** to the owner of the call which will then make the transition to the ***mod_ep_successful*** state.

If the modification was not successful, the network will send a negative **mod_ep RESPONSE** to the owner. The ***op_status*** field will give an indication of the reason for the failure. Upon receipt of the negative **RESPONSE** the owner will make the transition to the ***mod_ep_failed*** state.

7.4.15announce_change

Synopsis:

This operation informs an endpoint about a change in a call. The REQUEST phase goes from network to client.

Data:

- ***op_type*** - **announce_change**
- ***phase*** - defines which type of message this is.

  phase ∈ {REQUEST}
- ***msg_id*** - message identifier, unique to a client endpoint
- ***call_id*** - call identifier of the call that was modified.
- ***num_cons*** - specifies the number of connections being specified in the **announce_change**.
- ***options_size*** - size of the following ***options*** field. If options_size is 0, then there is no ***options*** field present. The ***options_size*** field is always present.
- ***options*** - variable length field for introduction of new options and parameters. Currently there are no options defined.

For the connection specified in the **announce_change REQUEST**, a connection specification is given which consists of the following information:

- ***con_id*** - connection identifier, uniquely identifies a connection within a call
- ***con_type*** - specifies what type of connection this is. This field is a 3-tuple.

  con_type = < channel_type ∈ {VP, VC},
  bw_ type ∈ {STATIC, DYNAMIC},
  qos ∈ {HIGH, MEDIUM, LOW}>
- ***con_perm*** - default permission specification for this connection

  con_perm = < receive = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}>,
  transmit = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}> >
- ***ep_perm*** - default endpoint permission specification for this connection.

  ep_perm = < receive = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}>,
  transmit = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}> >
- ***ep_map*** - specifies the UNI mapping to be used for the additional endpoint for this connection.

  ep_perm = < receive = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}>,
  transmit = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}>,
  echo ∈ {ON, OFF} >
- ***notify*** - specifies whether endpoints should be notified if this connection is changed via a **mod_con** operation (Section 7.4.13).

  notify ∈ {**YES**, **NO**}
- ***bw*** - bandwidth specification for the connection.

  bw = <peak, average, peak_burst_length>
- ***user_con_type*** - user description of the type of this connection. Not manipulated by the network, just passed end to end.

Message Formats:

announce_change REQUEST

00001111	00000000	msg_id (2)		reserved (2)		reserved (2)	
call_id(o_addr) (24)							
call_id(lcid) (2)		unused (2)		user_call_type (4)			
unused (1)	unused (1)	mon (1)	unused(1)	unused(1)	reserved (1)	reserved (2)	
reserved (4)				reserved (2)		num_cons (2)	
con_id (2)		con_type (1)	notify (1)	bw (12)			
user_con_type (2)		unused (2)		con_perm(1)	unused (1)	unused (1)	unused (1)
reserved (1)	unused(1)	unused (2)		reserved (1)	unused (1)	unused (2)	
• • •							
con_id (2)		con_type (1)	notify (1)	bw (12)			
user_con_type (2)		unused (2)		con_perm(1)	unused (1)	unused (1)	unused (1)
reserved (1)	unused(1)	unused (2)		reserved (1)	unused (1)	unused (2)	
options_size (4)				options (options_size)			

Operation:

This operation, **announce_change**, is used to inform an endpoint that a call that it is participating in, has been modified. The modification could have resulted from a **mod_con** or **mod_call** operation.

Figure 69. Endpoint being notified of a modification to a call

States:
0: active_call

If the **announce_change** is a result of a **mod_call** operation, then **num_cons** will be 0, there will be no connection specifications listed. The only change being reported will be the new **user_call_type**.

If the announce_change is a result of a **mod_con**, num_cons will indicate how many connections were modified and a connection specification for each will detail the changes.

7.4.16open_con_invite

Synopsis:

This operation invites an endpoint to join a connection which has been added to a call it is already a part of. It is triggered when the owner requests an ***open_con*** operation. The REQUEST phase goes from network to owner.

Data:

- ***op_type*** - **open_con_invite**
- ***phase*** - defines which type of message this is.

  phase ∈ {REQUEST, RESPONSE, CONFIRMATION}
- ***msg_id*** - message identifier, unique to a client endpoint
- ***call_id*** - call identifier of the call to which connections are being added.
- ***op_status* -** describes the reason for the failure of the operation.(Section 7.3.23)

  op_status* = <*call_status_bit ∈ {CALL_SPEC_OK, CALL_SPEC_ERROR},
  connection_status_bit ∈ {CON_SPEC_OK, CON_SPEC_ERROR},
  status ∈ {OK, REFUSED,
  BAD_PHASE_EXP_RESPONSE,
  TIMEOUT, ERROR, NEGOTIATING}>
- ***num_cons*** - specifies the number of connections being specified in the **invite**.
- ***ep_id*** - endpoint identifier for the endpoint being invited.
- ***options_size*** - size of the following ***options*** field. If options_size is 0, then there is no ***options*** field present. The ***options_size*** field is always present.
- ***options*** - variable length field for introduction of new options and parameters. Currently there are no options defined.

For each connection specified in the **open_con_invite REQ**, a connection specification is given which consists of the following information:

- ***con_id*** - connection identifier, uniquely identifies a connection within a call
- ***con_type*** - specifies what type of connection this is. This field is a 3-tuple.

  con_type = < channel_type ∈ {VP, VC},
  bw_ type ∈ {STATIC, DYNAMIC},
  qos ∈ {HIGH, MEDIUM, LOW}>
- ***con_perm*** - default permission specification for this connection

  con_perm = < receive = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}>,
  transmit = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}> >
- ***ep_perm*** - default endpoint permission specification for this connection.

  ep_perm = < receive = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}>,
  transmit = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}> >
- ***ep_map*** - specifies the UNI mapping to be used for the additional endpoint for this connection.

  ep_perm = < receive = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}>,
  transmit = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}>,
  echo ∈ {ON, OFF} >

- ***notify*** - specifies whether endpoints should be notified if this connection is changed via a **mod_con** operation (Section 7.4.13).

    notify ∈ {**YES**, **NO**}

- ***bw*** - bandwidth specification for the connection.

    bw = <peak, average, peak_burst_length>

- ***user_con_type*** - user description of the type of this connection. Not manipulated by the network, just passed end to end.
- ***trans_vpi*** - transmit virtual path identifier for the new endpoint on this connection
- ***trans_vci*** - transmit virtual channel identifier for the new endpoint on this connection.
- ***rcv_vpi*** - receive virtual path identifier for the new endpoint on this connection
- ***rcv_vci*** - receive virtual channel identifier for the new endpoint on this connection.
- ***con_status*** - this field is used to indicate what was potentially wrong with specific connection specifications. (Section 7.3.24)

    con_status ∈ {OK,
    DUP_CON_ID, BAD_CON_TYPE, BAD_NOTIFY,
    NO_AVAIL_VCI, RCV_VCI_IN_USE, TRANS_VCI_IN_USE,
    RCV_VCI_RESERVED, RCV_VCI_NOT_SUPPORTED,
    TRANS_VCI_RESERVED, TRANS_VCI_NOT_SUPPORTED,
    NO_AVAIL_VPI, RCV_VPI_IN_USE, TRANS_VPI_IN_USE,
    RCV_VPI_RESERVED, RCV_VPI_NOT_SUPPORTED,
    TRANS_VPI_RESERVED, TRANS_VPI_NOT_SUPPORTED,
    ILL_PERM_CHANGE, BAD_EP_PERM, BAD_BW}

Message Formats:

open_con_invite REQUEST

00010000	00000000	msg_id (2)	reserved (2)	reserved (2)	
call_id(o_addr) (24)					
call_id(lcid) (2)	unused (2)	reserved (2)	num_cons (2)		
r_addr (24)					
con_id (2)	con_type (1)	notify (1)	bw (12)		
user_con_type (2)	unused (2)	con_perm(1)	ep_perm (1)	ep_map (1)	unused (1)
reserved (1)	trans_vpi(1)	trans_vci (2)	reserved (1)	rcv_vpi (1)	rcv_vci (2)
• • •					
con_id (2)	con_type (1)	notify (1)	bw (12)		
user_con_type (2)	unused (2)	con_perm(1)	ep_perm (1)	ep_map (1)	unused (1)
reserved (1)	trans_vpi(1)	trans_vci (2)	reserved (1)	rcv_vpi (1)	rcv_vci (2)
options_size (4)	options (options_size)				

Message Formats (cont):

open_con_invite RESPONSE

<table>
<tr><td>00010000</td><td>00000001</td><td colspan="2">msg_id (2)</td><td colspan="2">reserved (2)</td><td colspan="2">reserved (2)</td></tr>
<tr><td colspan="8" rowspan="3">call_id(o_addr) (24)</td></tr>
<tr></tr>
<tr></tr>
<tr><td colspan="2">call_id(lcid) (2)</td><td colspan="2">op_status (2)</td><td colspan="2">reserved (2)</td><td colspan="2">num_cons (2)</td></tr>
<tr><td colspan="2">con_id (2)</td><td>con_type (1)</td><td>notify (1)</td><td colspan="4" rowspan="2">bw (12)</td></tr>
<tr><td colspan="4"></td></tr>
<tr><td colspan="2">user_con_type (2)</td><td colspan="2">con_status (2)</td><td>con_perm(1)</td><td>ep_perm (1)</td><td>ep_map (1)</td><td>unused (1)</td></tr>
<tr><td>reserved (1)</td><td>trans_vpi(1)</td><td colspan="2">trans_vci (2)</td><td>reserved (1)</td><td>rcv_vpi (1)</td><td colspan="2">rcv_vci (2)</td></tr>
<tr><td colspan="8">•
•
•</td></tr>
<tr><td colspan="2">con_id (2)</td><td>con_type (1)</td><td>notify (1)</td><td colspan="4" rowspan="2">bw (12)</td></tr>
<tr><td colspan="4"></td></tr>
<tr><td colspan="2">user_con_type (2)</td><td colspan="2">con_status (2)</td><td>con_perm(1)</td><td>ep_perm (1)</td><td>ep_map (1)</td><td>unused (1)</td></tr>
<tr><td>reserved (1)</td><td>trans_vpi(1)</td><td colspan="2">trans_vci (2)</td><td>reserved (1)</td><td>rcv_vpi (1)</td><td colspan="2">rcv_vci (2)</td></tr>
<tr><td colspan="4">options_size (4)</td><td colspan="4">options (options_size)</td></tr>
</table>

Message Formats (cont):

open_con_invite CONFIRMATION

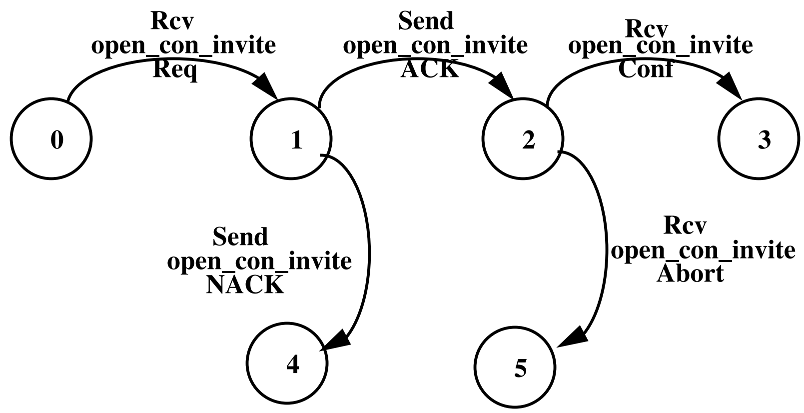
00010000	00000010	msg_id (2)		reserved (2)		reserved (2)	
call_id(o_addr) (24)							
call_id(lcid) (2)		op_status (2)		reserved (2)		num_cons (2)	
con_id (2)		con_type (1)	notify (1)	bw (12)			
user_con_type (2)		con_status (2)		con_perm(1)	ep_perm (1)	ep_map (1)	unused (1)
reserved (1)	trans_vpi(1)	trans_vci (2)		reserved (1)	rcv_vpi (1)	rcv_vci (2)	
⋮							
con_id (2)		con_type (1)	notify (1)	bw (12)			
user_con_type (2)		con_status (2)		con_perm(1)	ep_perm (1)	ep_map (1)	unused (1)
reserved (1)	trans_vpi(1)	trans_vci (2)		reserved (1)	rcv_vpi (1)	rcv_vci (2)	
options_size (4)				options (options_size)			

Operation:

An **open_con_invite** operation is triggered when the owner of the call issues an **open_con REQUEST**. The purpose of the **open_con_invite** operation is to ask participating endpoints whether they want to participate in the new connections.

Since the endpoint is already active in the call, the Client CMAP Agent is assumed to be in the *active_call* state when it receives the **open_con_invite REQUEST**.. Upon receiving the **REQUEST**, the Client CMAP Agent should

Figure 70. Endpoint being invited to join new connections in a call



States:

0: active_call
1: process_open_con_invite_request
2: acknowledged_open_con_invite
3: open_con_invite_success
4: open_con_invite_refused
5: open_con_invite_failed

make the transition to the *process_open_con_invite_request* state. The information about each new connection that is accessible is provided in the **REQUEST.** This gives the invited endpoint the information to decide whether it wants to join the new connections and whether it has the required resources.

A connection is inaccessible if its ***ep_perm*** is set as follows:

ep_perm = < receive = < perm = OFF, changeability = OFF>,

transmit = < perm = OFF, changeability = OFF>,

echo $\in$ {ON, OFF} >

Inaccessible connections are only listed in the **open_con_invite** to the owner. This would occur if the **modifiable** parameter of the call was set to **ON** and an endpoint other than the owner added a connection with the default permissions set as shown above. The **open_con** for such a connection would trigger an **open_con_invite** for only the owner.

If it does not want to join the new connections, the Client CMAP Agent should generate a negative **open_con_invite RESPONSE** (***op_status*** field of **REFUSED**) and make the transition back to the *no_call* state.

If it does want to join the new connection, the Client CMAP Agent should generate a positive **open_con_invite RESPONSE** (***op_status*** field of **OK**) and make the transition to the *acknowledged_open_con_invite* state.

If any of the VPI/VCI pairs specified in the **REQUEST** phase are not acceptable to the endpoint, it may change them in the acknowledgment phase. When the client wishes to make changes to the VPI/VCI pairs, it sets the ***op_status*** field to NEGOTIATING. If the network interface node does not accept these changes, the operation fails and the endpoint receives a negative **open_con_invite CONFIRMATION**. The Client CMAP Agent would then make the transition to the *open_con_invite_refused* state.

The endpoint can also modify its permissions, **ep_perm**, and notify parameter for the new connections. Changes to the permissions may trigger a **verify** operation (Section 7.4.7). If the owner does not accept the changes it requested, the operation fails and the endpoint receives a negative **open_con_invite** CONFIRMATION. The Client CMAP Agent would then make the transition back to the *no_call* state.

7.4.17mod_ep_invite

Synopsis:

This operation invites an endpoint to accept changes that have been made to connections it is already a part of. It is triggered when the owner requests an ***mod_ep*** operation. The REQUEST phase goes from network to owner.

Data:

- ***op_type*** - **mod_ep**
- ***phase*** - defines which type of message this is.

  phase ∈ {REQUEST, RESPONSE}
- ***msg_id*** - message identifier, unique to a client endpoint
- ***call_id*** - call identifier of the call for which the specified endpoint is being modified.
- ***ep_id*** - endpoint identifier for the endpoint being modified.
- ***num_cons*** - number of connections in the call being modified
- ***op_status*** **-** describes the reason for the failure of the operation.(Section 7.3.23)

  op_status* = <*call_status_bit ∈ {CALL_SPEC_OK**,** CALL_SPEC_ERROR},
  connection_status_bit ∈ {CON_SPEC_OK**,** CON_SPEC_ERROR},
  status ∈ {OK, REFUSED, TIMEOUT, ERROR, NEGOTIATING}>
- ***options_size*** - size of the following ***options*** field. If options_size is 0, then there is no ***options*** field present. The ***options_size*** field is always present.
- ***options*** - variable length field for introduction of new options and parameters. Currently there are no options defined.

For each connection specified in the **invite REQ**, a connection specification is given which consists of the following information:

- ***con_id*** - connection identifier, uniquely identifies a connection within a call
- ***con_type*** - specifies what type of connection this is. This field is a 3-tuple.

  con_type = < channel_type ∈ {VP, VC},
  bw_ type ∈ {STATIC, DYNAMIC},
  qos ∈ {HIGH, MEDIUM, LOW}>
- ***con_perm*** - default permission specification for this connection

  con_perm = < receive = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}>,
  transmit = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}> >
- ***ep_perm*** - default endpoint permission specification for this connection.

  ep_perm = < receive = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}>,
  transmit = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}> >
- ***ep_map*** - specifies the UNI mapping to be used for the additional endpoint for this connection.

  ep_perm = < receive = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}>,
  transmit = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}>,
  echo ∈ {ON, OFF} >
- ***notify*** - specifies whether endpoints should be notified if this connection is changed via a **mod_con** operation (Section 7.4.13).

notify ∈ {**YES**, **NO**}

- ***bw*** - bandwidth specification for the connection.

  bw = <peak, average, peak_burst_length>

- ***user_con_type*** - user description of the type of this connection. Not manipulated by the network, just passed end to end.
- ***trans_vpi*** - transmit virtual path identifier for the new endpoint on this connection
- ***trans_vci*** - transmit virtual channel identifier for the new endpoint on this connection.
- ***rcv_vpi*** - receive virtual path identifier for the new endpoint on this connection
- ***rcv_vci*** - receive virtual channel identifier for the new endpoint on this connection.
- ***con_status*** - this field is used to indicate what was potentially wrong with specific connection specifications. (Section 7.3.24)

  con_status ∈ {OK,
  DUP_CON_ID, BAD_CON_TYPE, BAD_NOTIFY,
  NO_AVAIL_VCI, RCV_VCI_IN_USE, TRANS_VCI_IN_USE,
  RCV_VCI_RESERVED, RCV_VCI_NOT_SUPPORTED,
  TRANS_VCI_RESERVED, TRANS_VCI_NOT_SUPPORTED,
  NO_AVAIL_VPI, RCV_VPI_IN_USE, TRANS_VPI_IN_USE,
  RCV_VPI_RESERVED, RCV_VPI_NOT_SUPPORTED,
  TRANS_VPI_RESERVED, TRANS_VPI_NOT_SUPPORTED,
  ILL_PERM_CHANGE, BAD_EP_PERM, BAD_BW}

Message Formats:

mod_ep_invite REQUEST

00010001	00000000	msg_id (2)		reserved (2)		reserved (2)	
call_id(o_addr) (24)							
call_id(lcid) (2)		unused (2)		ep_id (2)		num_cons (2)	
r_addr (24)							
con_id (2)		con_type (1)	notify (1)	bw (12)			
user_con_type (2)		unused (2)		con_perm(1)	ep_perm (1)	ep_map (1)	unused (1)
reserved (1)	trans_vpi(1)	trans_vci (2)		reserved (1)	rcv_vpi (1)	rcv_vci (2)	
⋮							
con_id (2)		con_type (1)	notify (1)	bw (12)			
user_con_type (2)		unused (2)		con_perm(1)	ep_perm (1)	ep_map (1)	unused (1)
reserved (1)	trans_vpi(1)	trans_vci (2)		reserved (1)	rcv_vpi (1)	rcv_vci (2)	
options_size (4)				options (options_size)			

Message Formats (cont):

mod_ep_invite RESPONSE

00010001	00000001	msg_id (2)		reserved (2)		reserved (2)	
call_id(o_addr) (24)							
call_id(lcid) (2)		op_status (2)		ep_id (2)		num_cons (2)	
con_id (2)		con_type (1)	notify (1)	bw (12)			
user_con_type (2)		con_status (2)		con_perm(1)	ep_perm (1)	ep_map (1)	unused (1)
reserved (1)	trans_vpi(1)	trans_vci (2)		reserved (1)	rcv_vpi (1)	rcv_vci (2)	
• • •							
con_id (2)		con_type (1)	notify (1)	bw (12)			
user_con_type (2)		con_status (2)		con_perm(1)	ep_perm (1)	ep_map (1)	unused (1)
reserved (1)	trans_vpi(1)	trans_vci (2)		reserved (1)	rcv_vpi (1)	rcv_vci (2)	
options_size (4)				options (options_size)			

Message Formats (cont):

mod_ep_invite CONFIRMATION

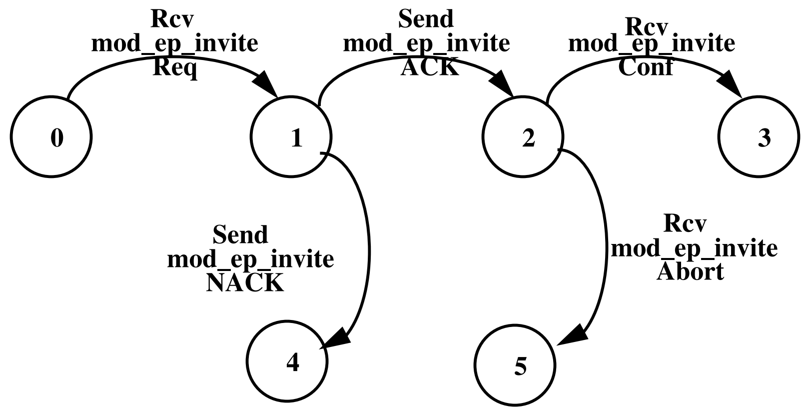
00010001	00000010	msg_id (2)		reserved (2)		reserved (2)	
call_id(o_addr) (24)							
call_id(lcid) (2)		op_status (2)		ep_id (2)		num_cons (2)	
con_id (2)		con_type (1)	notify (1)	bw (12)			
user_con_type (2)		con_status (2)		con_perm(1)	ep_perm (1)	ep_map (1)	unused (1)
reserved (1)	trans_vpi(1)	trans_vci (2)		reserved (1)	rcv_vpi (1)	rcv_vci (2)	
⋮							
con_id (2)		con_type (1)	notify (1)	bw (12)			
user_con_type (2)		con_status (2)		con_perm(1)	ep_perm (1)	ep_map (1)	unused (1)
reserved (1)	trans_vpi(1)	trans_vci (2)		reserved (1)	rcv_vpi (1)	rcv_vci (2)	
options_size (4)				options (options_size)			

Operation:

An **mod_ep_invite** operation is triggered when the owner of the call issues an **mod_ep REQUEST**. The purpose of the **mod_ep_invite** operation is to inform the specified endpoint of the changes requested by the owner and allow the endpoint to negotiated new VPI/VCI pairs when necessary.

Since the endpoint is already active in the call, the Client CMAP Agent is assumed to be in the *active_call* state when it receives the **mod_ep_invite REQUEST**.. Upon receiving the **REQUEST**, the Client CMAP Agent should

Figure 71. Endpoint being informed of changes to its connection permissions



States:
0: active_call
1: process_mod_ep_invite_request
2: acknowledged_mod_ep_invite
3: mod_ep_invite_success
4: mod_ep_invite_refused
5: mod_ep_invite_failed

make the transition to the *process_mod_ep_invite_request* state. The information about each of the new connections is provided in the **REQUEST.** This gives the invited endpoint the information to decide whether it wants to join the new connections and whether it has the required resources.

If it does not want to participate in the connections as specified in the **mod_ep_invite REQUEST**, the Client CMAP Agent should generate a negative **mod_ep_invite RESPONSE** (***op_status*** field of **REFUSED**) and make the transition to the *mod_ep_invite_refused* state. This will cause the network to remove the endpoint from participation in the connections specified in the **REQUEST**.

If it does want to participate in the connections as specified, the Client CMAP Agent should generate a positive **mod_ep_invite RESPONSE** (***op_status*** field of **OK**) and make the transition to the *acknowledged_mod_ep_invite* state.

If any of the VPI/VCI pairs specified in the **REQUEST** phase are not acceptable to the endpoint, it may change them in the acknowledgment phase. When the client wishes to make changes to the VPI/VCI pairs, it sets the ***op_status*** field to NEGOTIATING. If the network interface node does not accept these changes, the operation fails and the endpoint receives a negative **mod_ep_invite CONFIRMATION**. This will cause the network to remove the endpoint from participation in the connections specified in the **REQUEST** The Client CMAP Agent would then make the transition to the *mod_ep_invite_failed* state.

The endpoint can also modify its permissions, **ep_perm**, and notify parameter for the new connections. Changes to the permissions may trigger a **verify** operation (Section 7.4.7). If the owner does not accept the changes it requested or if the permissions were not changeable, the operation fails and the endpoint receives a negative **mod_ep_invite** CONFIRMATION. This will cause the network to remove the endpoint from participation in the connections specified in the **REQUEST** The Client CMAP Agent would then make the transition to the *mod_ep_invite_failed* state.

7.4.18network_reset

Synopsis:

This operation informs a client termination that its network interface node has been reset and all call information concerning it has been cleared. The REQUEST phase goes from network to client.

Data:

- ***op_type*** - **network_reset**
- ***phase*** - defines which type of message this is.

  phase ∈ {REQUEST}
- ***msg_id*** - message identifier, unique to a client endpoint
- ***call_id*** - call identifier, data in this field is not used, just included to complete header.
- ***options_size*** - size of the following ***options*** field. If options_size is 0, then there is no ***options*** field present. The ***options_size*** field is always present.
- ***options*** - variable length field for introduction of new options and parameters. Currently there are no options defined.

Message Formats:

network_reset REQUEST

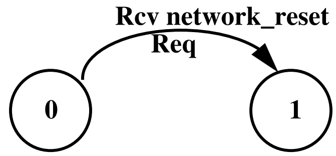
<table>
<tr><td>00010010</td><td>00000000</td><td>msg_id (2)</td><td>reserved (2)</td><td>reserved (2)</td></tr>
<tr><td colspan="5">call_id(o_addr) (24)</td></tr>
<tr><td colspan="2">call_id(lcid) (2)</td><td>unused (2)</td><td colspan="2">reserved (4)</td></tr>
<tr><td colspan="3">options_size (4)</td><td colspan="2">options (options_size)</td></tr>
</table>

Operation:

This operation, **network_reset**, is used to inform a client termination that its network interface node has been reset and all information concerning calls it was involved in has been cleared. The ***msg_id*** and ***call_id*** fields have no meaning in this message, but are included to fill the basic message structure.

The Client CMAP Agent may receive this message while in any state. Upon receipt of the network_reset REQUEST it should perform whatever rebuilding procedures it has to recreate the calls that were lost.

Figure 72. Client being informed of a network interface node reset



States:
0: any state
1: no_call

7.4.19client_reset

Synopsis:

This operation informs a network interface node that a client termination has been reset and all call information concerning it has been cleared. The REQUEST phase goes from client to network.

Data:

- ***op_type*** - **client_reset**
- ***phase*** - defines which type of message this is.

  phase ∈ {REQUEST, RESPONSE}
- ***msg_id*** - message identifier, unique to a client endpoint
- ***call_id*** - call identifier, data in this field is not used, just included to complete header.
- ***op_status*** **-** describes the reason for the failure of the operation.(Section 7.3.23)

  op_status = <call_status_bit ∈ {CALL_SPEC_OK, CALL_SPEC_ERROR},

  connection_status_bit ∈ {CON_SPEC_OK, CON_SPEC_ERROR},

  status ∈ {OK}>
- ***options_size*** - size of the following ***options*** field. If options_size is 0, then there is no ***options*** field present. The ***options_size*** field is always present.
- ***options*** - variable length field for introduction of new options and parameters. Currently there are no options defined.

Message Formats:

client_reset REQUEST

<table>
<tr><td>00010011</td><td>00000000</td><td>msg_id (2)</td><td>reserved (2)</td><td>reserved (2)</td></tr>
<tr><td colspan="5">call_id(o_addr) (24)</td></tr>
<tr><td colspan="2">call_id(lcid) (2)</td><td>unused (2)</td><td colspan="2">reserved (4)</td></tr>
<tr><td colspan="3">options_size (4)</td><td colspan="2">options (options_size)</td></tr>
</table>

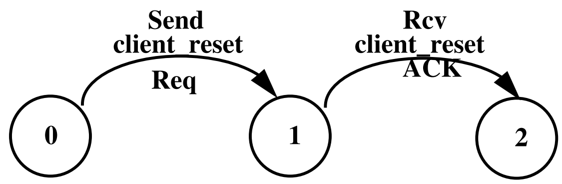
client_reset RESPONSE

<table>
<tr><td>00010011</td><td>00000001</td><td>msg_id (2)</td><td>reserved (2)</td><td>reserved (2)</td></tr>
<tr><td colspan="5">call_id(o_addr) (24)</td></tr>
<tr><td colspan="2">call_id(lcid) (2)</td><td>op_status (2)</td><td colspan="2">reserved (4)</td></tr>
<tr><td colspan="3">options_size (4)</td><td colspan="2">options (options_size)</td></tr>
</table>

Operation:

This operation, **client_reset**, is used to inform a network interface node that a client termination has been reset and all information concerning calls it was involved in has been cleared. The ***msg_id*** and ***call_id*** fields have no meaning in this message but are included to fill the basic message structure.

Figure 73. Client informing network of a client reset



States:
0: any state
1: processing_client_reset
2: no_call

The network interface node will return an acknowledgment indicating to the client that its reset was performed and that it may begin performing new connection management operations.

7.4.20trace_call

Synopsis:

This operation requests that a call be traced. The REQUEST phase goes from client to network.

Data:

- ***op_type*** - **trace_call**
- ***phase*** - defines which type of message this is.

  phase ∈ {REQUEST, RESPONSE}
- ***msg_id*** - message identifier, unique to a client endpoint
- ***call_id*** - call identifier of the traced call
- ***user_call_type*** - user description of the type of this call. Not manipulated by the network, just passed end to end.
- ***call_type*** - specifies what type of call this is.

  call_type ∈ {POINT_TO_POINT, MULTIPOINT}
- ***acc*** - specifies whether endpoints may freely join the call.

  acc ∈ {OPEN, VERIFY, CLOSED}
- ***mon*** - specifies who should be notified when endpoints join, drop or are modified in the call.

  mon = <owner ∈ {ON, OFF}, transmitters ∈ {ON, OFF}, all ∈ {ON, OFF} >
- ***modifiability*** - specifies whether endpoints other than the owner may add connections to this call.

  modifiability ∈ {ON, OFF}
- ***priority*** - specifies a priority level for this call.

  priority ∈ {NORMAL, PREEMPT, OVERRIDE}
- ***num_cons*** - specifies the number of connections being specified in the **trace_call RESPONSE**.
- ***own_ep_id*** - endpoint identifier for the owner of the call.
- ***num_ep*** **-** specifies the number of endpoints being specified in the **trace_call RESPONSE**
- ***op_status*** **-** describes the reason for the failure of the operation.(Section 7.3.23)

  op_status = <call_status_bit ∈ ***{***CALL_SPEC_***OK,*** CALL_SPEC_***ERROR},***

  connection_status_bit ∈ ***{***CON_SPEC_***OK,*** CON_SPEC_***ERROR},***

  status ∈ {OK, DUP_CALL_ID, DUP_LCID, BAD_CALL_ID_ADDR,
  INSUFF_BW, VERIFY_REFUSED,
  EP_REFUSED, DUP_MSG_ID, BAD_MSG_ID,
  BAD_PHASE_EXP_REQUEST,
  BAD_ACC, BAD_MON, BAD_CALL_TYPE,
  BAD_NUM_CONS, BAD_PRIORITY, BAD_MODIFIABLE,
  BAD_NUM_EP, BAD_EP_ADDR, TIMEOUT}>
- ***options_size*** - size of the following ***options*** field. If options_size is 0, then there is no ***options*** field present. The ***options_size*** field is always present.
- ***options*** - variable length field for introduction of new options and parameters. Currently there are no options defined.

For each endpoint specified in the **trace_call RESPONSE**, an endpoint specification is given which consists of the following information:

- ***ep_addr*** - endpoint address of an endpoint active in the call.
- ***ep_id*** - endpoint identifier for the endpoint.

For each connection specified in the **trace_call RESPONSE**, a connection specification is given which consists of the following information:

- ***con_id*** - connection identifier, uniquely identifies a connection within a call
- ***con_type*** - specifies what type of connection this is. This field is a 3-tuple.

  con_type = < channel_type $\in$ {VP, VC},
  bw_ type $\in$ {STATIC, DYNAMIC},
  qos $\in$ {HIGH, MEDIUM, LOW}>

- ***con_perm*** - default permission specification for this connection

  con_perm = < receive = < perm $\in$ {ON, HOLD, OFF}, changeability $\in$ {ON, VERIFY, OFF}>,
  transmit = < perm $\in$ {ON, HOLD, OFF}, changeability $\in$ {ON, VERIFY, OFF}> >

- ***notify*** - specifies whether endpoints should be notified if this connection is changed via a **mod_con** operation (Section 7.4.13).

  notify $\in$ {**YES**, **NO**}

- ***bw*** - bandwidth specification for the connection.

  bw = <peak, average, peak_burst_length>

- ***user_con_type*** - user description of the type of this connection. Not manipulated by the network, just passed end to end.

Message Formats:

trace_call REQUEST

00010100	**00000001**	**msg_id (2)**	**reserved (2)**	**reserved (2)**
call_id(o_addr) (24)				
call_id(lcid) (2)		**unused (2)**	**reserved (4)**	
options_size (4)			**options (options_size)**	

Message Formats (cont):

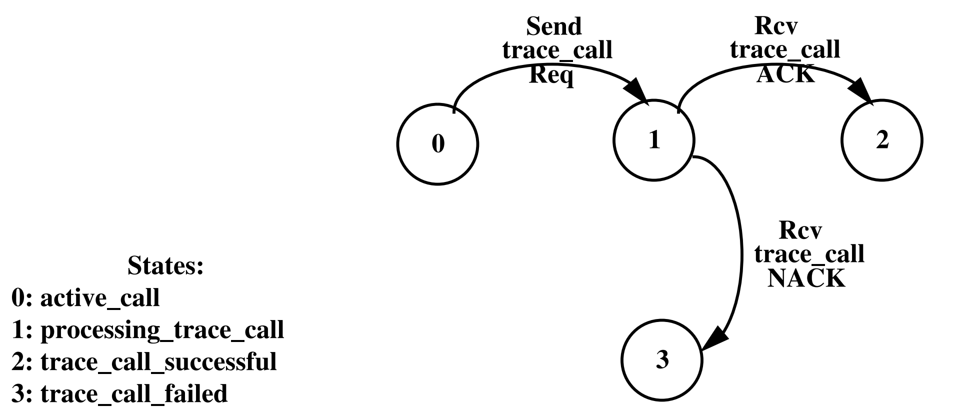
trace_call RESPONSE

00010100	00000001	msg_id (2)		reserved (2)		reserved (2)	
call_id(o_addr) (24)							
call_id(lcid) (2)		op_status (2)		user_call_type (4)			
call_type (1)	acc (1)	mon (1)	modifiable(1)	priority(1)	reserved (1)	reserved (2)	
own_ep_id (2)		num_cons (2)		reserved (2)		num_ep(2)	
ep_addr (24)							
ep_id (2)		reserved (2)		reserved (4)			
• • •							
ep_addr (24)							
ep_id (2)		reserved (2)		reserved (4)			
con_id (2)		con_type (1)	notify (1)	bw (12)			
user_con_type (2)		con_status (2)		con_perm(1)	unused (1)	unused (1)	unused (1)
reserved (1)	unused(1)	unused (2)		reserved (1)	unused (1)	unused (2)	
• • •							
con_id (2)		con_type (1)	notify (1)	bw (12)			
user_con_type (2)		con_status (2)		con_perm(1)	unused (1)	unused (1)	unused (1)
reserved (1)	unused(1)	unused (2)		reserved (1)	unused (1)	unused (2)	
options_size (4)				options_size (options_size)			

Operation:

This operation allows a client to request information from the network about a call. Connection information and a list of the endpoints participating in the call is returned.

Figure 74. Request for trace of call state



The **trace_call** and **trace_ep** operations can be combined to obtain all the available information about a call and its participants.

NB: It is unclear at this time what kind of security will be necessary for such an operation. It is possible that it will only be allowed for the owner of a call.

7.4.21 trace_ep

Synopsis:

This operation requests that a call be traced. The REQUEST phase goes from client to network.

Data:

- ***op_type*** - **trace_ep**
- ***phase*** - defines which type of message this is.

  phase ∈ {REQUEST, RESPONSE}
- ***msg_id*** - message identifier, unique to a client endpoint
- ***call_id*** - call identifier of the traced call
- ***user_call_type*** - user description of the type of this call. Not manipulated by the network, just passed end to end.
- ***call_type*** - specifies what type of call this is.

  call_type ∈ {POINT_TO_POINT, MULTIPOINT}
- ***acc*** - specifies whether endpoints may freely join the call.

  acc ∈ {OPEN, VERIFY, CLOSED}
- ***mon*** - specifies who should be notified when endpoints join, drop or are modified in the call.

  mon = <owner ∈ {ON, OFF}, transmitters ∈ {ON, OFF}, all ∈ {ON, OFF} >
- ***modifiability*** - specifies whether endpoints other than the owner may add connections to this call.

  modifiability ∈ {ON, OFF}
- ***priority*** - specifies a priority level for this call.

  priority ∈ {NORMAL, PREEMPT, OVERRIDE}
- ***num_cons*** - specifies the number of connections being specified in the **trace_ep RESPONSE**.
- ***own_ep_id*** - endpoint identifier for the owner of the call.
- ***num_ep*** **-** specifies the number of endpoints being specified in the **trace_ep RESPONSE**
- ***op_status*** **-** describes the reason for the failure of the operation.(Section 7.3.23)

  op_status = <call_status_bit ∈ ***{***CALL_SPEC_***OK,*** CALL_SPEC_***ERROR},***

  connection_status_bit ∈ ***{***CON_SPEC_***OK,*** CON_SPEC_***ERROR},***

  status ∈ {OK, DUP_CALL_ID, DUP_LCID, BAD_CALL_ID_ADDR,
  INSUFF_BW, VERIFY_REFUSED,
  EP_REFUSED, DUP_MSG_ID, BAD_MSG_ID,
  BAD_PHASE_EXP_REQUEST,
  BAD_ACC, BAD_MON, BAD_CALL_TYPE,
  BAD_NUM_CONS, BAD_PRIORITY, BAD_MODIFIABLE,
  BAD_NUM_EP, BAD_EP_ADDR, TIMEOUT}>
- ***options_size*** - size of the following ***options*** field. If options_size is 0, then there is no ***options*** field present. The ***options_size*** field is always present.
- ***options*** - variable length field for introduction of new options and parameters. Currently there are no options defined.

For each endpoint specified in the **trace_ep RESPONSE**, an endpoint specification is given which consists of the following information:

- ***ep_addr*** - endpoint address of an endpoint active in the call.
- ***ep_id*** - endpoint identifier for the endpoint.

For each connection specified in the **trace_ep RESPONSE**, a connection specification is given which consists of the following information:

- ***con_id*** - connection identifier, uniquely identifies a connection within a call
- ***con_type*** - specifies what type of connection this is. This field is a 3-tuple.

  con_type = < channel_type ∈ {VP, VC},
  bw_ type ∈ {STATIC, DYNAMIC},
  qos ∈ {HIGH, MEDIUM, LOW}>
- ***con_perm*** - default permission specification for this connection

  con_perm = < receive = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}>,
  transmit = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}> >
- ***ep_perm*** - default endpoint permission specification for this connection.

  ep_perm = < receive = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}>,
  transmit = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}> >
- ***ep_map*** - specifies the UNI mapping to be used for the additional endpoint for this connection.

  ep_perm = < receive = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}>,
  transmit = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}>,
  echo ∈ {ON, OFF} >
- ***notify*** - specifies whether endpoints should be notified if this connection is changed via a **mod_con** operation (Section 7.4.13).

  notify ∈ {**YES**, **NO**}
- ***bw*** - bandwidth specification for the connection.

  bw = <peak, average, peak_burst_length>
- ***user_con_type*** - user description of the type of this connection. Not manipulated by the network, just passed end to end.

Message Formats:

trace_ep REQUEST

<table>
<tr><td>00010101</td><td>00000001</td><td>msg_id (2)</td><td>reserved (2)</td><td>reserved (2)</td></tr>
<tr><td colspan="5">call_id(o_addr) (24)</td></tr>
<tr><td colspan="2">call_id(lcid) (2)</td><td>unused (2)</td><td colspan="2">reserved (4)</td></tr>
<tr><td colspan="5">ep_addr (24)</td></tr>
<tr><td colspan="2">ep_id (2)</td><td>reserved (2)</td><td colspan="2">reserved (4)</td></tr>
<tr><td colspan="3">options_size (4)</td><td colspan="2">options (options_size)</td></tr>
</table>

Message Formats (cont):

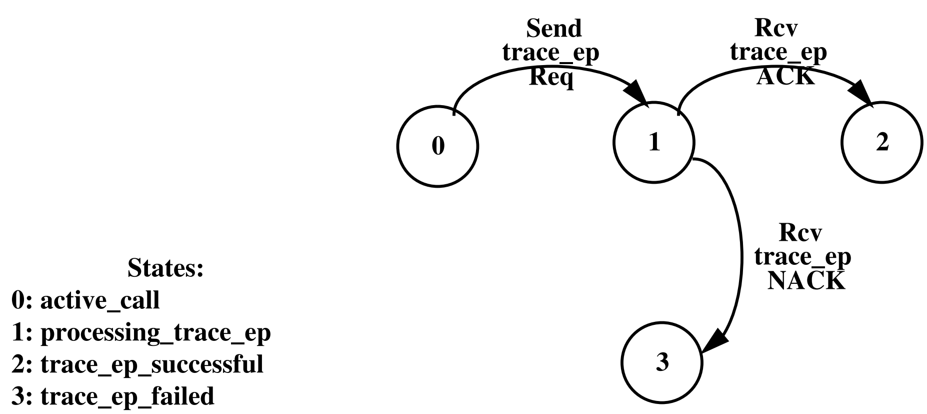
trace_ep RESPONSE

<table>
<tr><td>00010101</td><td>00000001</td><td colspan="2">msg_id (2)</td><td colspan="2">reserved (2)</td><td colspan="2">reserved (2)</td></tr>
<tr><td colspan="8">call_id(o_addr) (24)</td></tr>
<tr><td colspan="2">call_id(lcid) (2)</td><td colspan="2">op_status (2)</td><td colspan="2">reserved (2)</td><td colspan="2">num_cons (2)</td></tr>
<tr><td colspan="8">ep_addr (24)</td></tr>
<tr><td colspan="2">ep_id (2)</td><td colspan="2">reserved (2)</td><td colspan="4">reserved (4)</td></tr>
<tr><td colspan="2">con_id (2)</td><td>con_type (1)</td><td>notify (1)</td><td colspan="4" rowspan="2">bw (12)</td></tr>
<tr><td colspan="4"></td></tr>
<tr><td colspan="2">user_con_type (2)</td><td colspan="2">con_status (2)</td><td>con_perm(1)</td><td>ep_perm (1)</td><td>ep_map (1)</td><td>unused (1)</td></tr>
<tr><td>reserved (1)</td><td>trans_vpi(1)</td><td colspan="2">trans_vci (2)</td><td>reserved (1)</td><td>rcv_vpi (1)</td><td colspan="2">rcv_vci (2)</td></tr>
<tr><td colspan="8">•
•
•</td></tr>
<tr><td colspan="2">con_id (2)</td><td>con_type (1)</td><td>notify (1)</td><td colspan="4" rowspan="2">bw (12)</td></tr>
<tr><td colspan="4"></td></tr>
<tr><td colspan="2">user_con_type (2)</td><td colspan="2">con_status (2)</td><td>con_perm(1)</td><td>ep_perm (1)</td><td>ep_map (1)</td><td>unused (1)</td></tr>
<tr><td>reserved (1)</td><td>trans_vpi(1)</td><td colspan="2">trans_vci (2)</td><td>reserved (1)</td><td>rcv_vpi (1)</td><td colspan="2">rcv_vci (2)</td></tr>
<tr><td colspan="4">options_size (4)</td><td colspan="4">options_size (options_size)</td></tr>
</table>

Operation:

This operation allows a client to request information from the network about an endpoints participation in a call. The connection information, including the ep_perm and VPI/VCI pairs, as it pertains to the specified endpoint is provided for each connection in the call.

Figure 75. Request for trace of call state



The **trace_call** and **trace_ep** operations can be combined to obtain all the available information about a call and its participants.

NB: It is unclear at this time what kind of security will be necessary for such an operation. It is possible that it will only be allowed for the owner of a call.

7.4.22change_owner

Synopsis:

This operation requests that the owner of a call be changed. This may only be requested by the owner of the call. The new owner must already be a part of the call. The REQUEST phase goes from owner to network.

Data:

- ***op_type*** - **change_owner**
- ***phase*** - defines which type of message this is.

  phase ∈ {REQUEST, RESPONSE}
- ***msg_id*** - message identifier, unique to a client endpoint
- ***call_id*** - call identifier of the call for which ***change_owner*** is being performed.
- ***ep_addr*** - address of endpoint to be made the new owner of the call.
- ***ep_id*** - endpoint identifier for the new owner of the call.
- ***new_id*** - should a new call_id be generated

  new_id ∈ {YES, NO}
- ***new_call_id*** -the new unique call identifier for the call (generated by the network).
- ***op_status* -** describes the reason for the failure of the operation.(Section 7.3.23)

  op_status* = <*call_status_bit ∈ {CALL_SPEC_OK, CALL_SPEC_ERROR},
  connection_status_bit ∈ {CON_SPEC_OK, CON_SPEC_ERROR},
  status ∈ {OK, BAD_CALL_ID_ADDR,
  EP_REFUSED, DUP_MSG_ID, BAD_MSG_ID,
  BAD_PHASE_EXP_REQUEST,
  BAD_EP_ADDR, TIMEOUT}>
- ***options_size*** - size of the following ***options*** field. If options_size is 0, then there is no ***options*** field present. The ***options_size*** field is always present.
- ***options*** - variable length field for introduction of new options and parameters. Currently there are no options defined.

Message Formats:

change_owner REQUEST

00010110	00000000	msg_id (2)	reserved (2)	reserved (2)
call_id(o_addr) (24)				
call_id(lcid) (2)	unused (2)	new_id (1)	reserved (1)	ep_id (2)
ep_addr (24)				
options_size (4)		options (options_size)		

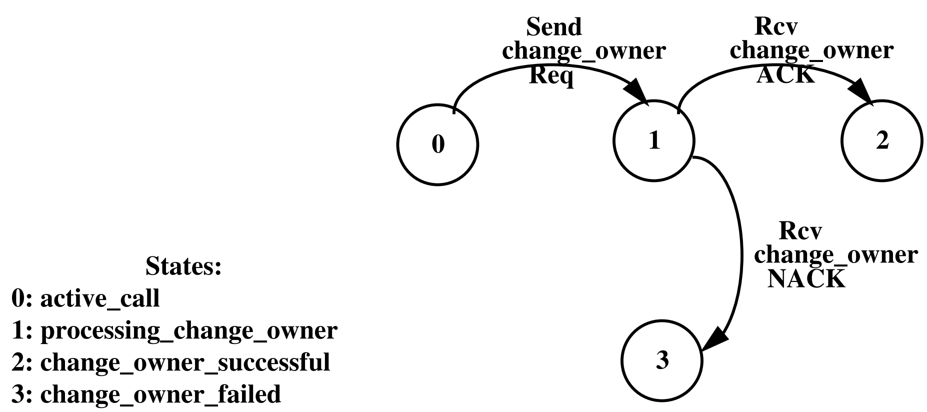
change_owner RESPONSE

00010110	00000001	msg_id (2)	reserved (2)	reserved (2)
call_id(o_addr) (24)				
call_id(lcid) (2)	op_status (2)			
new_call_id(o_addr) (24)				
		new_call_id(lcid) (2)	reserved (2)	
options_size (4)		options (options_size)		

Operation:

The owner of a call uses the **change_owner** operation to transfer ownership to another endpoint. This operation

Figure 76. Current owner requesting call be changed to have a new owner



triggers an **invite_owner** operation inviting the specified endpoint to become the new owner of the call.

The initiator of the change_owner operation has to option to specify whether a new call_id should be generated for the call. The new call_id is returned in the **RESPONSE** phase of the operation. If no new_id was requested then the new_call_id field is identical to the call_id field.

7.4.23invite_owner

Synopsis:

This operation invites an endpoint in a call to become the new owner of that call. The REQUEST goes from network to client.

Data:

- ***op_type*** - **invite_owner**
- ***phase*** - defines which type of message this is.

  phase ∈ {REQUEST, RESPONSE, CONFIRMATION}
- ***msg_id*** - message identifier, unique to a client endpoint
- ***call_id*** - call identifier of the call for which ***invite_owner*** is to be performed.
- ***user_call_type*** - user description of the type of this call. Not manipulated by the network, just passed end to end.
- ***call_type*** - specifies what type of call this is.

  call_type ∈ {POINT_TO_POINT, MULTIPOINT}
- ***acc*** - specifies whether endpoints may freely join the call.

  acc ∈ {OPEN, VERIFY, CLOSED}
- ***mon*** - specifies who should be notified when endpoints join, drop or are modified in the call.

  mon = <owner ∈ {ON, OFF}, transmitters ∈ {ON, OFF}, all ∈ {ON, OFF} >
- ***modifiability*** - specifies whether endpoints other than the owner may add connections to this call.

  modifiability ∈ {ON, OFF}
- ***priority*** - specifies a priority level for this call.

  priority ∈ {NORMAL, PREEMPT, OVERRIDE}
- ***num_cons*** - specifies the number of connections being specified in the **invite_owner RESPONSE**.
- ***new_call_id*** -the new unique call identifier for the call (generated by the network).
- ***op_status* -** describes the reason for the failure of the operation.(Section 7.3.23)

  op_status* = <*call_status_bit ∈ ***{***CALL_SPEC_***OK,*** CALL_SPEC_***ERROR},***
  connection_status_bit ∈ ***{***CON_SPEC_***OK,*** CON_SPEC_***ERROR},***
  status ∈ {OK, DUP_CALL_ID, DUP_LCID, BAD_CALL_ID_ADDR,
  INSUFF_BW, VERIFY_REFUSED,
  EP_REFUSED, DUP_MSG_ID, BAD_MSG_ID,
  BAD_PHASE_EXP_RESPONSE,
  BAD_ACC, BAD_MON, BAD_CALL_TYPE,
  BAD_NUM_CONS, BAD_PRIORITY, BAD_MODIFIABLE,
  TIMEOUT}>
- ***options_size*** - size of the following ***options*** field. If options_size is 0, then there is no ***options*** field present. The ***options_size*** field is always present.
- ***options*** - variable length field for introduction of new options and parameters. Currently there are no options defined.

For each endpoint specified in the **invite_owner RESPONSE**, an endpoint specification is given which consists of the following information:

- ***ep_addr*** - endpoint address of an endpoint active in the call.

- ***ep_id*** - endpoint identifier for the endpoint.

For each connection specified in the **invite_owner RESPONSE**, a connection specification is given which consists of the following information:

- ***con_id*** - connection identifier, uniquely identifies a connection within a call
- ***con_type*** - specifies what type of connection this is. This field is a 3-tuple.

$$\textbf{\textit{con\_type}} = < \text{channel\_type} \in \{\text{VP, VC}\},$$
$$\text{bw\_ type} \in \{\text{STATIC, DYNAMIC}\},$$
$$\text{qos} \in \{\text{HIGH, MEDIUM, LOW}\}>$$

- ***con_perm*** - default permission specification for this connection

$$\textbf{\textit{con\_perm}} = < \text{receive} = < \text{perm} \in \{\text{ON, HOLD, OFF}\}, \text{changeability} \in \{\text{ON, VERIFY, OFF}\}>,$$
$$\text{transmit} = < \text{perm} \in \{\text{ON, HOLD, OFF}\}, \text{changeability} \in \{\text{ON, VERIFY, OFF}\}> >$$

- ***notify*** - specifies whether endpoints should be notified if this connection is changed via a **mod_con** operation (Section 7.4.13).

$$\textbf{\textit{notify}} \in \{\textbf{YES}, \textbf{NO}\}$$

- ***bw*** - bandwidth specification for the connection.

$$\textbf{\textit{bw}} = <\text{peak, average, peak\_burst\_length}>$$

- ***user_con_type*** - user description of the type of this connection. Not manipulated by the network, just passed end to end.

Message Formats:

invite_owner REQUEST

00010111	00000000	msg_id (2)		reserved (2)		reserved (2)	
call_id(o_addr) (24)							
call_id(lcid) (2)		unused (2)		user_call_type (4)			
call_type (1)	acc (1)	mon (1)	modifiable(1)	priority(1)	reserved (1)	reserved (2)	
new_call_id(o_addr) (24)							
new_call_id(lcid) (2)		reserved (2)		reserved (2)		num_cons (2)	
con_id (2)		con_type (1)	notify (1)	bw (12)			
user_con_type (2)		unused (2)		con_perm(1)	unused (1)	unused (1)	unused (1)
reserved (1)	unused(1)	unused (2)		reserved (1)	unused (1)	unused (2)	
• • •							
con_id (2)		con_type (1)	notify (1)	bw (12)			
user_con_type (2)		unused (2)		con_perm(1)	unused (1)	unused (1)	unused (1)
reserved (1)	unused(1)	unused (2)		reserved (1)	unused (1)	unused (2)	
options_size (4)				options (options_size)			

Message Formats (cont):

invite_owner RESPONSE

00010111	**00000001**	**msg_id (2)**	
call_id(o_addr) (24)			
call_id(lcid) (2)		**op_status (2)**	
new_call_id(o_addr) (24)			
		new_call_id(lcid) (2)	**reserved (2)**
options_size (4)		**options_size (4)**	

invite_owner CONFIRMATION

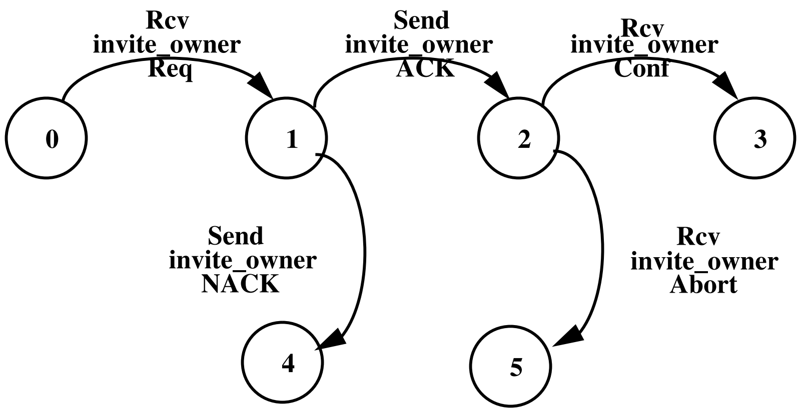
00010111	**00000010**	**msg_id (2)**	**reserved (2)**	**reserved (2)**
call_id(o_addr) (24)				
call_id(lcid) (2)		**op_status (2)**	**reserved (4)**	
options_size (4)			**options_size (4)**	

Operation:

After the owner of a call issues a **change_owner** REQUEST, an **invite_owner** operation is triggered to ask the specified endpoint to become the new owner of the call. The request phase of the operation specifies the new call identifier, which was generated by the network. The messages in this operation always use the old call identifier in the call_id field. After the confirm phase, the old call identifier will no longer be recognized and the specified endpoint becomes the new owner of the call. The successful completion of this operation triggers an **announce_owner** operation to notify all the other endpoints in the call that the owner, and the call identifier of the call has been changed.

.

Figure 77. Endpoint being invited to become new owner of a call



States:

0: active_call
1: process_invite_owner_request
2: acknowledged_invite_owner
3: invite_owner_success
4: invite_owner_refused
5: invite_owner_failed

7.4.24announce_owner

Synopsis:

This operation notifies a participant in a call that the owner of the call has changed and there is a new call identifier. The REQUEST phase goes from network to client.

Data:

- ***op_type*** - **announce_owner**
- ***phase*** - defines which type of message this is.

  phase $\in$ {REQUEST}
- ***msg_id*** - message identifier, unique to a client endpoint
- ***call_id*** - the old call identifier of the call for which ***announce_owner*** is performed.
- ***new_call_id*** - new call identifier for the call.
- ***options_size*** - size of the following ***options*** field. If options_size is 0, then there is no ***options*** field present. The ***options_size*** field is always present.
- ***options*** - variable length field for introduction of new options and parameters. Currently there are no options defined.

Message Formats:

announce_owner REQUEST

00011000	**00000000**	**msg_id (2)**	**reserved (2)**	**reserved (2)**
call_id(o_addr) (24)				
call_id(lcid) (2)		**unused (2)**	**new_call_id(o_addr) (24)**	
			new_call_id(lcid) (2)	**reserved (2)**
options_size (4)			**options (options_size)**	

Operation:

The owner of a call has been changed. This operation notifies the endpoints in the call of the new call identifier. From now on, the call must be referred to by the new call identifier. **What happens to outstanding REQUESTs that used the old call_id???**

Figure 78. Endpoint being notified of a new owner for a call

States:
0: active_call

8. CMAP Implementations

An implementation of CMAP is distinguished from the general CMAP specification. The design of CMAP described in Sections 3 through 7 was based on a general model of the ATM protocol and fast packet switching networks. A network for which a CMAP implementation is designed may not be able to support all of the capabilities described. For this reason we allow an implementation of CMAP to vary from the generalized model. The network is still considered a CMAP network as long as it conforms to at least the minimum specification described in Appendix D. CMAP, in its most general form, supports a dynamic, *n*-way multipoint connection oriented ATM network. In more restricted forms, it supports a reduced set where, for instance, a CMAP implementation might only support point-to-point or one-to-*n* multipoint connections. In either case, the network is still considered a CMAP network since the basic CMAP operations and handshake procedures are used. By making CMAP as general as possible but allowing it to have implementation subsets, we provide a single call management protocol that can be layered over a diverse set of networks.

If all CMAP options are implemented and all abstractions of the call model are supported in a CMAP implementation, then we refer to this as *complete CMAP*. We have also defined a *minimal CMAP* that specifies the minimum capabilities that the network must support in order to be considered a CMAP network. These two CMAP implementations are contrasted in Table 9. The minimal CMAP is on the left of the table and the complete CMAP is on the right, with the functional effect of each CMAP option adjacent to them respectively. Table 9 is not an exhaustive list of CMAP options but just highlights the major options that have maximum impact of network functionality. The complete list of options for each is listed in Appendix D.

TABLE 9. Sample List of CMAP Implementation Options and the Functional Effect of Supporting Them.

Minimal CMAP Network	Minimal CMAP Functional Effect	Option	Complete CMAP Functional Effect	Complete CMAP Network
VP's not supported, must support VC's	Clients Nacked when requesting VP connections.	**Virtual Path & Channel Connections**	Full VP capability supported.	VP's and VC's fully supported
Point-to-Point only	The client must specify a POINT-TO-POINT call type and adding a third client will be Nacked.	**Multicast**	Any number of clients may access a call/connection. Clients may still choose the POINT-TO-POINT call type if they desire.	Fully supported
CMAP will collect BW data (peak, average, burst) and may ignore it.	Little, if any, BW blocking may occur. Relies solely on statistical multiplexing of data, may have large variations in cell pipe performance.	**Resource Management Enforcement**	BW blocking may occur but connections that are established will be guaranteed their performance within certain statistical boundaries.	Collects BW data, contention for shared resources between clients is resolved and sets up hardware to enforce the connection BW behavior.
Meta-signaling channel required, but may not support more than one.	Multidrop client terminations, if supported at all, may have to share a meta-signaling channel.	**Meta-Signaling Channel**	Clients on a multidrop channel may request a private signaling channel	One default channel required with the capability to request more.
May not be supported.	Network may respond to one client only.	**Multidrop Client Terminations at the UNI**	Network must respond to as many clients as exist on a single UNI termination.	Fully supported.

9. Future Directions in CMAP

To be written. Ideas include a transactional grouping of multiple CMAP operations and a generalization of the modifiability call parameter to grant degrees of call and connection manipulation capabilities to endpoints.

Appendix A: References

The reference list contains items that pertain to the area of fast packet switching and broadband networks, possibly of interest to the CMAP user or implementor. Many of the references apply directly to the issue of call management and are cited in this document. Other references are for general background purposes only.

[1] H. Ahmadi, W.E. Denzel, C.A. Murphy, and E. Port. “A High-Performance Switch Fabric for Integrated Circuit and Packet Switching.” In IEEE Infocom ‘88: Proceedings of the Seventh Annual Joint Conference of the IEEE Computer and Communications Societies, pages 9-18, March 1988.

[2] H. Ahmadi and W. E. Denzel. “A Survey of Modern High-Performance Switching Techniques.” In IEEE Journal on Selected Areas in Communications, 7(7):1091-1103, September 1989.

[3] ANSI T1S1 Technical Sub-Committee. Broadband Aspects of ISDN Baseline Document. T1S1.5/90-001, June 1990.

[4] R. Ballart and Y.C. Ching. “SONET: Now Its the Standard Optical Network.” In IEEE Communications Magazine, 27(3):8-15, March 1989.

[5] Bell Communications Research. Generic System Requirements in Support of Switched Multi-Megabit Data Service. Technical Advisory TA-TSY-000772, Issue 3, October 1989.

[6] P.A. Bernstein, V. Hadzilacos, and N. Goodman. Concurrency Control and Recovery in Database Systems. Addison-Wesley, 1987.

[7] R.G. Bubenik and J.S. Turner. “Performance of a Broadcast Packet Switch.” In IEEE Transactions on Communications, 37(1):60-69, January 1989.

[8] R. G. Bubenik, J. D. DeHart and M. E. Gaddis. “Multipoint Connection Management in High Speed Networks.” In IEEE Infocom ‘91: Proceedings of the Tenth Annual Joint Conference of the IEEE Computer and Communications Societies, pages 59-68, April 1991.

[9] R. G. Bubenik, M. E. Gaddis and J. D. DeHart. “A Strategy for Layering IP over ATM”. Washington University Applied Research Laboratory, Working Note 91-01, Version 1.1, April 1991.

[10] R.G. Bubenik, M.E. Gaddis, and J.D. DeHart. “Virtual Paths and Virtual Channels.” To appear in IEEE Infocom ‘92: Proceedings of the Eleventh Annual Joint Conference of the IEEE Computer and Communications Societies, May 1992.

[11] R. G. Bubenik. “BPN Reliable Datagram Protocol”. Washington University Applied Research Laboratory Working Note 91-11, in progress, June 1991.

[12] J. Burgin and D. Dorman. “Broadband ISDN Resource Management: The Role of Virtual Paths.” In IEEE Communications Magazine, 29(9):44-48, September 1991.

[13] CCITT. Blue Book, volume II, fascicle II.2, “Telephone network and ISDN—Operation, numbering, routing, and mobile service,” Recommendations E.100--E.300, Geneva, Switzerland, 1989.

[14] CCITT. Recommendations Drafted by Working Party XVIII/8 (General B-ISDN Aspects) to be Approved in 1992, Study Group XVIII—Report R 34, December 1991.

[15] CCITT Recommendation Q.931 (I.451), ISDN User-Network Interface Layer 3 Specification, Geneva, 1985.

[16] D.R. Cheriton and W. Zwaenepoel. “Distributed Process Groups in the V Kernel.” In Transactions on Computer Systems, 3(2):77-107, May 1985.

[17] R. Colella, E. Gardner and R. Callon. “Guidelines for OSI NSAP Allocation in the Internet.” INTERNET DRAFT, Networking Group, March 1, 1991.

[18] D. Comer. Internetworking With TCP/IP Principles, Protocols, and Architecture. Prentice Hall, 1988.

[19] J.P. Coudreuse and M Servel. “PRELUDE: An Asynchronous Time-Division Switched Network.” In ICC ‘87: Proceedings of the IEEE International Conference on Communications, pages 69-773, June 1987.

[20] Jr. R. Cox. “Overview of the Washington University Fast Packet Project”. Washington University, Applied Research Laboratory Working Note 89-02, September 1989.

[21] J. R. Cox and J. S. Turner. "Project Zeus Design and Application of Fast Packet Campus Networks". Washington University, Department of Computer Science Technical Report 91-45, July 1991.

[22] G.E. Daddis, Jr. and H.C. Torng. "A Taxonomy of Broadband Integrated Switching Architectures." In IEEE Communications Magazine, 27(5):32-42, May 1989.

[23] S.E. Deering. "Multicast Routing in Internetworks and Extended LANs." In Proceedings of the SIGCOMM '88 Symposium: Communications Architectures & Protocols, pages 55-64, August 1988.

[24] K.Y. Eng, M.G. Hluchyj, and Y.S. Yeh. "Multicast and Broadcast Services in a Knockout Packet Switch." In IEEE Infocom '88: Proceedings of the Seventh Annual Joint Conference of the IEEE Computer and Communications Societies, pages 29-34, March 1988.

[25] H.C. Folts. "Procedures for Circuit-Switched Service in Synchronous Public Data Networks." In IEEE Transactions on Communications, 28(4):489-496, April 1980.

[26] M. E. Gaddis. "ATM-TAP: Patent Disclosure Statement". Washington University, Applied Research Laboratory Working Note 90-12, Version 1.2, May 1990.

[27] M. E. Gaddis, R.G. Bubenik, and J.D. DeHart. "Connection Management for a Prototype Fast Packet ATM B-ISDN Network." In Proceedings of the National Communications Forum, vol. 44, pp. 601-608, October 8-10, 1990.

[28] M. E. Gaddis, R.G. Bubenik, and J.D. DeHart. "A Call Model for Multipoint Communications in Switched Networks." submitted for publication to ICC '92, Chicago, Illinois, June 1992.

[29] J.N Giacopelli, W.D. Sincoskie, and M. Littlewood. "Sunshine: A High Performance Self-Routing Broadband Packet Switch Architecture." In Proceedings of the International Switching Symposium, Volume 3, pages 123-129, May 1990.

[30] W.M. Harman and C.F. Newman. "ISDN Protocols for Connection Control." In IEEE Journal on Selected Areas in Communications, 7(7):1034-1042, September 1989.

[31] K. Haserodt and J.S. Turner. "An Architecture for Connection Management in a Broadcast Packet Network." Washington University, Department of Computer Science, Technical Report-WUCS-87-03, 1987.

[32] M.G. Hluchyj and M.J. Karol. "Queueing in Space-Division Packet Switching." In IEEE Infocom '88: Proceedings of the Seventh Annual Joint Conference of the IEEE Computer and Communications Societies, pages 334-343, March 1988.

[33] A. Huang and S. Knauer. "Starlite: a Wideband Digital Switch." In Proceedings of Globecom 84, pages 121-125, December 1984.

[34] J. Hui. "A Broadband Packet Switch for Multi-Rate Services." In ICC '87: Proceedings of the IEEE International Conference on Communications, pages 782-788, June 1987.

[35] K. Iguchi, H. Takeo, S. Amemiya, and K. Tezuka. "Subscriber Access Scheme for Broadband ISDN." In ICC '90: Proceedings of the IEEE International Conference on Communications, pages 663-669, April 1990.

[36] A.R. Jacob. A Survey of Fast Packet Switches. Computer Communication Review, 20(1):54-64, January 1990.

[37] Y. Kato, T. Shimoe, K. Hajikano, and K. Murakami. "Experimental Broadband ATM Switching System." In Proceedings of Globecom 88, pages 1288-1292, December 1988.

[38] H.S. Kim and A. Leon-Garcia. "A Self-Routing Multistage Switching Network for Broadband ISDN." In IEEE Journal on Selected Areas in Communications, 8(3):459-466, April 1990.

[39] J.C. Kohli, D.S. Biring, and G.L. Raya. "Emerging Broadband Packet-Switch Technology in Integrated Information Networks." In IEEE Network, 2(6):37-38,47-51, November 1988.

[40] T.R. La Porta and M. Schwartz. "Architectures, Features, and Implementation of High-Speed Transport Protocols." In IEEE Network, 4(2):14-22, May 1991.

[41] T.T. Lee, R. Boorstyn, and E. Arthurs. "The Architecture of a Multicast Broadband Packet Switch." In IEEE Infocom '88: Proceedings of the Seventh Annual Joint Conference of the IEEE Computer and Communications Societies, pages 1-8, March 1988.

[42] T. Lyon. “Simple and Efficient Adaptation Layer” ANSI T1S1.5 proposal for type 5 AAL by Sun Microsystems, Inc., August, 12-16, 1991.

[43] S.E. Minzer. “Broadband ISDN and Asynchronous Transfer Mode (ATM).” In IEEE Communications Magazine, 27(9):17-24, September 1989.

[44] S.E. Minzer and D.R. Spears. “New Directions in Signaling for Broadband ISDN.” In IEEE Communications Magazine, 27(2):6-14, February 1989.

[45] S.E. Minzer. “A Signaling Prototype for Complex Multimedia Services.” In IEEE Journal on Selected Areas in Communications, 9(9):1383-1394, December 1991.

[46] J.E.B. Moss. Nested Transactions: An Approach to Reliable Distributed Computing. MIT Press, 1985.

[47] C.H. Papadimitriou. The Theory of Concurrency Control. Computer Science Press, 1986.

[48] G.M. Parulkar, J.S. Turner. Towards a Framework for High Speed Communication in a Heterogeneous Networking Environment. In IEEE Infocom ‘89: Proceedings of the Eighth Annual Joint Conference of the IEEE Computer and Communications Societies, pages 655-667, April 1989.

[49] G. M. Parulkar. “The Next Generation of Internetworking”. ACM SIGCOMM Computer Communications Review. vol. 20, no. 1, New York, NY, pp. 18-43, January, 1990.

[50] F.E. Ross. “An Overview of FDDI: The Fiber Distributed Data Interface.” In IEEE Journal on Selected Areas in Communications, 7(7):1043-1051, September 1989.

[51] A. Rybczynski. “X.25 Interface and End-to-End Virtual Circuit Service Characteristics.” In IEEE Transactions on Communications, 28(4):500-510, April 1980.

[52] R.M. Sanders. The Xpress Transfer Protocol (XTP)—A Tutorial. Computer Networks Laboratory, Department of Computer Science, University of Virginia, January 15, 1990.

[53] J.S. Stacey, T. Pham, and J. Chiou. “Modeling Call Control for Distributed Applications in Telephony.” In IEEE Network, 4(6):14-22, November 1990.

[54] H. Suzuki, H. Nagano, T. Suzuki, T. Takeuchi, and S. Iwasaki. “Output-buffer Switch Architecture for Asynchronous Transfer Mode.” In ICC ‘89: Proceedings of the IEEE International Conference on Communications, pages 99-103, 1989.

[55] H. Suzuki, T. Murase, S. Sato, and T. Takeuchi. “A Burst Traffic Control Strategy for ATM Networks.” Submitted for publication (conference unknown).

[56] A.S. Tanenbaum. Computer Networks. Prentice-Hall, 1981.

[57] S.C. Tu and W.H. Leung. “Multicast Connection-Oriented Packet Switching Networks.” In Proceedings of the International Communications Conference, volume 2, pages 495-501, April 1990.

[58] J. S. Turner, “Fast Packet Switching System”, U.S. Patent 4 494 230, January 15, 1985.

[59] J.S. Turner. “New Directions in Communications.” In IEEE Communications Magazine, 24(10):8-15, October 1986.

[60] J.S. Turner. “Design of an Integrated Services Packet Network.” In IEEE Transactions on Communications, 4(8):1373-1380, November 1986.

[61] J.S. Turner. “Design of a Broadcast Packet Switching Network.” In IEEE Transactions on Communications, 36(6):734-743, June 1988.

[62] J. S. Turner. “A Proposed Management and Congestion Control Scheme for Multicast ATM Networks.” Washington University, Computer and Communication Research Center Technical Report 91-01, May 1991.

[63] XTP® Protocol Definition, Revision 3.5. Protocol Engines Incorporated, Technical Report PEI 90-120, September 10, 1990.

[64] Y.S. Yeh, M.G. Hluchyj, and A.S. Acampora. “The Knockout Switch: A Simple, Modular Architecture for Performance Packet Switching.” In International Switching Symposium, volume 3, pages 801-808, March 1987.

Appendix B: CMAP Operation Field Values

op_type

0x0	no operation
0x1	open_call
0x2	close_call
0x3	add_ep
0x4	drop_ep
0x5	drop_from_call
0x6	invite
0x7	open_con
0x8	close_con
0x9	verify
0xa	mod_call
0xb	mod_con
0xc	mod_ep
0xd	announce_ep
0xe	announce_drop
0xf	announce_change
0x10	transfer
0x11	change_owner
0x12	invite_owner
0x13	announce_owner
0x14	network_reset
0x15	client_reset
0x16	open_con_invite
0x17	mod_ep_invite
0x18	trace_call
0x19	trace_ep
others	reserved for future use

phase

0x0	REQUEST
0x1	RESPONSE
0x2	CONFIRMATION

op_status: call_status_bit

0x0	CALL_SPEC_OK
0x1	CALL_SPEC_ERROR

op_status: connection_status_bit

0x0	CON_SPEC_OK
0x1	CON_SPEC_ERROR

op_status: status

0x0	OK
0x1	DUP_CALL_ID
0x2	DUP_LCID

0x3	BAD_CALL_ID_ADDR
0x4	INSUFF_BW
0x5	VERIFY_REFUSED
0x6	EP_REFUSED
0x7	DUP_MSG_ID
0x8	BAD_MSG_ID
0x9	BAD_PHASE_EXP_REQUEST
0xa	BAD_PHASE_EXP_RESPONSE
0xb	BAD_PHASE_EXP_CONFIRMATION
0xc	BAD_ACC
0xd	BAD_MON
0xe	BAD_CALL_TYPE
0xf	BAD_NUM_CONS
0x10	BAD_PRIORITY
0x11	BAD_MODIFIABLE
0x12	BAD_NUM_EP
0x13	BAD_EP_ADDR
0x14	NEGOTIATING
0x15	TIMEOUT
others	reserved for future use

con_status

0x0	OK
0x1	DUP_CON_ID
0x2	BAD_CON_TYPE
0x3	BAD_NOTIFY
0x4	NO_AVAIL_VCI
0x5	RCV_VCI_IN_USE
0x6	RCV_VCI_RESERVED
0x7	RCV_VCI_NOT_SUPPORTED
0x8	TRANS_VCI_IN_USE
0x9	TRANS_VCI_RESERVED
0xa	TRANS_VCI_NOT_SUPPORTED
0xb	NO_AVAIL_VPI
0xc	RCV_VPI_IN_USE
0xd	RCV_VPI_RESERVED
0xe	RCV_VPI_NOT_SUPPORTED
0xf	TRANS_VPI_IN_USE
0x10	TRANS_VPI_RESERVED
0x11	TRANS_VPI_NOT_SUPPORTED
0x12	BAD_CON_PERM
0x13	BAD_EP_PERM
0x14	BAD_CRTR_PERM
0x15	BAD_BW
others	reserved for future use

accessibility

0x0	CLOSED
0x1	OPEN
0x2	VERIFY
others	reserved for future use

priority

0x0	NORMAL
0x1	PREEMPT
0x2	OVERRIDE
others	reserved for future use

monitor (owner, transmitter, all)

0x0	OFF
0x1	ON

call_type

0x0	MULTIPOINT
0x1	RESTRICTED_POINT_TO_POINT
others	reserved for future use

modifiability

0x0	OFF
0x1	ON

no_invite

0x0	NO
0x1	YES

notify

0x0	NO
0x1	YES

con_type: channel_type

0x0	VP (Virtual Path)
0x1	VC (Virtual Channel)
others	reserved for future use

con_type: bw_type

0x0	DYNAMIC
0x1	STATIC
others	reserved for future use

con_type: qos

0x0	HIGH
0x1	MEDIUM
0x2	LOW

permission: perm (transmit, receive, echo)

0x0	OFF
0x1	ON

permission: changeability (transmit, receive)

0x0	OFF
0x1	VERIFY
0x2	ON

client address: addr_type

0x0	undefined
0x1	IP
0x2	ISDN_E_164
0x3	NSAP
others	reserved for future use

Appendix C: CMAP Error Codes For *op_status* and *con_status* Fields

C.1 Status codes for the *op_status* field

status = OK

- Description: No errors were detected. The operation was successful.
- Recommended Action:
- Operations: all

status = DUP_CALL_ID

- Description: The ***call_id*** specified in an **open_call REQUEST** is already in use.
- Recommended Action: Select a new ***call_id*** and resend the **open_call REQUEST**.
- Operations: **open_call**

status = DUP_LCID

- Description: The ***lcid*** portion of the ***call_id*** specified in an **open_call REQUEST** is already in use.
- Recommended Action: Select a new ***lcid*** and resend the ***open_call*** **REQUEST**.
- Operations: **open_call**

status = BAD_CALL_ID_ADDR

- Description: The owner address portion of the ***call_id*** specified in an **open_call REQUEST** was not a proper endpoint address or did not match the address of the client that sent the **REQUEST**.
- Recommended Action: Check the address given and correct it. Resend the **open_call REQUEST** with the correct address.
- Operations: **open_call**

status = INSUFF_BW

- Description: There was not enough bandwidth in the network to perform the requested operation.
- Recommended Action: The client should check that it has enough bandwidth on its access link. If it does not have the bandwidth to support the new request, it should drop or drop out of some calls to free some bandwidth. If the access link has the bandwidth to support the new request dropping or dropping out of some calls may free enough bandwidth in the network. Otherwise the client should wait and try the request at a later time.
- Operations: **open_call**, **add_ep**, **open_con**, **mod_con**

status = VERIFY_REFUSED

- Description: The **REQUEST** phase requested an endpoint permission that required the owner to verify the permission. The owner refused to allow the requested permission.
- Recommended Action: The client should request a different permission or contact the owner of the call to discuss the call set up.
- Operations: **open_call**, **add_ep**, **mod_ep**

status = EP_REFUSED

- Description: The endpoint that was invited to join the call refused to join.
- Recommended Action: The client should try again at a later time.
- Operations: **open_call**, **add_ep**

status = DUP_MSG_ID

- Description: The ***msg_id*** given in the **REQUEST** is still active from a previous **REQUEST**
- Recommended Action: The client should generate a new ***msg_id*** and resend the **REQUEST**.

- Operations: all

status = BAD_MSG_ID

- Description: An illegal or unrecognized ***msg_id*** was used. A ***msg_id*** generated by a client should always have the highest order bit be 0. A network generated ***msg_id*** will have the highest order bit set to 1.
- Recommended Action: Generate a new ***msg_id*** making sure that the highest order bit is 0 and then resend the **REQUEST**.
- Operations: all

status = BAD_PHASE_EXP_REQUEST

- Description: An undefined value was found in the ***phase*** field. Based on the ***msg_id*** supplied a **REQUEST** was expected.
- Recommended Action: Generate the message again and resend.
- Operations: all

status = BAD_PHASE_EXP_RESPONSE

- Description: An undefined value was found in the ***phase*** field. Based on the ***msg_id*** supplied a **RESPONSE** was expected.
- Recommended Action: Generate the message again and resend.
- Operations: all

status = BAD_PHASE_EXP_CONFIRMATION

- Description: An undefined value was found in the ***phase*** field. Based on the ***msg_id*** supplied a **CONFIRMATION** was expected.
- Recommended Action: Generate the message again and resend.
- Operations: all

status = BAD_ACC

- Description: An undefined value was found in the ***acc*** field.
- Recommended Action: Correct the value and resend the **REQUEST**.
- Operations: **open_call**, **mod_call**

status = BAD_MON

- Description: An undefined value was found in the ***mon*** field.
- Recommended Action: Correct the value and resend the **REQUEST**.
- Operations: **open_call**, **mod_call**

status = BAD_CALL_TYPE

- Description: An undefined value was found in the ***call_type*** field.
- Recommended Action: Correct the value and resend the **REQUEST**.
- Operations: **open_call**

status = BAD_NUM_CONS

- Description: The number of connections specified in the ***num_cons*** field was out of range. The range of values if [0, MAX_CONS], where MAX_CONS is implementation dependent.
- Recommended Action: Redefine the call, possibly splitting the call into multiple calls and resend the **open_call REQUEST**.
- Operations: **open_call**

status = BAD_PRIORITY

- Description: An undefined value was found in the ***priority*** field.
- Recommended Action: Correct the value and resend the **REQUEST**.
- Operations: **open_call**, **mod_call**

status = BAD_MODIFIABLE

- Description: An undefined value was found in the ***modifiability*** field.
- Recommended Action: Correct the value and resend the **REQUEST**.
- Operations: **open_call**, **mod_call**

status = BAD_NUM_EP

- Description: The number of endpoints specified in the ***num_ep*** field was out of range. The range of values is [0, 1].
- Recommended Action: Resend the **open_call** with 0 or 1 endpoints. If more endpoints are desired submit a subsequent **add_ep REQUEST** for each one to be added.
- Operations: **open_call**

status = BAD_EP_ADDR

- Description: An unknown endpoint address was given.
- Recommended Action: Check the address to insure it is properly formed and it is a known address. Correct the address and resend the **REQUEST**
- Operations: **open_call**, **add_ep**,

status = TIMEOUT

- Description: A timeout occurred somewhere in the network.
- Recommended Action: Attempt the operation again. If it still fails with this ***status***, the client's network manager should be notified.
- Operations: all

C.2 Status codes for the *con_status* field

con_status = **OK**

- Description: No errors were detected in this connection specification.
- Recommended Action:
- Operations: all

con_status = DUP_CON_ID

- Description: The connection identifier specified in this connection specification is already in use for this call.
- Recommended Action:
- Operations: **open_call**, **open_con**

con_status = BAD_CON_TYPE

- Description: An undefined value was found in the ***con_type*** field.
- Recommended Action: Correct the value and resend the **REQUEST**
- Operations: **open_call**, **open_con**

con_status = BAD_NOTIFY

- Description: An undefined value was found in the ***notify*** field.
- Recommended Action: Correct the value and resend the **REQUEST**
- Operations: **open_call**, **open_con**

con_status = NO_AVAIL_VCI

- Description: There were no VCIs available to satisfy this **REQUEST**.
- Recommended Action:
- Operations: **open_call**, **open_con**, **add_ep**

con_status = RCV_VCI_IN_USE

- Description: The receive VCI requested is already in use.
- Recommended Action: Select another VCI and resend the **REQUEST**.
- Operations: **open_call**, **open_con**, **add_ep**

con_status = RCV_VCI_RESERVED

- Description: The receive VCI requested is reserved.
- Recommended Action: Select another VCI and resend the **REQUEST**. The reserved VCI should be marked as such in the client database so that it is not selected again.
- Operations: **open_call**, **open_con**, **add_ep**

con_status = RCV_VCI_NOT_SUPPORTED

- Description: The receive VCI requested is not supported. It is probably out of the range of VCIs supported by the hardware.
- Recommended Action: Check the range of VCIs supported and select a new VCI. Resend the **REQUEST**.
- Operations: **open_call**, **open_con**, **add_ep**

con_status = TRANS_VCI_IN_USE

- Description: The transmit VCI requested is already in use.
- Recommended Action: Select another VCI and resend the **REQUEST**.
- Operations: **open_call**, **open_con**, **add_ep**

con_status = TRANS_VCI_RESERVED

- Description: The transmit VCI requested is reserved.
- Recommended Action: Select another VCI and resend the **REQUEST**. The reserved VCI should be marked as such in the client database so that it is not selected again.
- Operations: **open_call**, **open_con**, **add_ep**

con_status = TRANS_VCI_NOT_SUPPORTED

- Description: The receive VCI requested is not supported. It is probably out of the range of VCIs supported by the hardware.
- Recommended Action: Check the range of VCIs supported and select a new VCI. Resend the **REQUEST**.
- Operations: **open_call**, **open_con**, **add_ep**

con_status = NO_AVAIL_VPI

- Description: There were no VPIs available to satisfy this **REQUEST**.
- Recommended Action:
- Operations: **open_call**, **open_con**, **add_ep**

con_status = RCV_VPI_IN_USE

- Description: The receive VPI requested is already in use.
- Recommended Action: Select another VPI and resend the **REQUEST**.

- Operations: **open_call**, **open_con**, **add_ep**

con_status = RCV_VPI_RESERVED

- Description: The receive VPI requested is reserved.
- Recommended Action: Select another VPI and resend the **REQUEST**. The reserved VPI should be marked as such in the client database so that it is not selected again.
- Operations: **open_call**, **open_con**, **add_ep**

con_status = RCV_VPI_NOT_SUPPORTED

- Description: The receive VCI requested is not supported. It is probably out of the range of VCIs supported by the hardware.
- Recommended Action: Check the range of VCIs supported and select a new VCI. Resend the **REQUEST**.
- Operations: **open_call**, **open_con**, **add_ep**

con_status = TRANS_VPI_IN_USE

- Description: The transmit VPI requested is already in use.
- Recommended Action: Select another VPI and resend the **REQUEST**.
- Operations: **open_call**, **open_con**, **add_ep**

con_status = TRANS_VPI_RESERVED

- Description: The transmit VPI requested is reserved.
- Recommended Action: Select another VPI and resend the **REQUEST**. The reserved VPI should be marked as such in the client database so that it is not selected again.
- Operations: **open_call**, **open_con**, **add_ep**

con_status = TRANS_VPI_NOT_SUPPORTED

- Description: The transmit VCI requested is not supported. It is probably out of the range of VCIs supported by the hardware.
- Recommended Action: Check the range of VCIs supported and select a new VCI. Resend the **REQUEST**.
- Operations: **open_call**, **open_con**, **add_ep**

con_status = BAD_CON_PERM

- Description: An undefined specification was given for the default permissions for this connection.
- Recommended Action: Correct the specification (see Sections 3.2.3, 7.2.4 and 7.3.15)
- Operations: **open_call**, **open_con**

con_status = BAD_EP_PERM

- Description: An undefined specification was given for the endpoint permissions for this connection.
- Recommended Action: Correct the specification so that it is of the form given below and resend the **REQUEST**.

  ep_perm = < receive = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}>,
  transmit = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}>,
  echo ∈ {ON, OFF} >

- Operations: **open_call**, **open_con**, **add_ep**

con_status = BAD_OWN_PERM

- Description: An undefined specification was given for the owner permissions for this connection.

- Recommended Action: Correct the specification so that it is of the form given below and resend the **REQUEST**.

  crtr_map = < receive = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}>,
  transmit = < perm ∈ {ON, HOLD, OFF}, changeability ∈ {ON, VERIFY, OFF}>,
  echo ∈ {ON, OFF} >

- Operations: **open_call**, **open_con**

con_status = BAD_BW

- Description: An illegal or impossible specification was given for the bandwidth for this connection.
- Recommended Action: The bandwidth should be specified in the form given below where the ***peak*** and ***average*** are specified in cells per second and the ***peak_burst_length*** is the number of cells to be sent during a peak burst. One possible cause for an illegal bandwidth specification is if the ***average*** is greater than the ***peak***. The client should check that the specification is correct and resend the **REQUEST**.

  ***bw* = <*peak*, *average*, *peak_burst_length*>**

- Operations: **open_call**, **open_con**, **add_ep**

Appendix D: Minimal CMAP Option List

To be written.

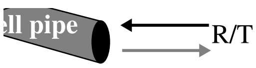
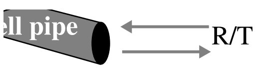
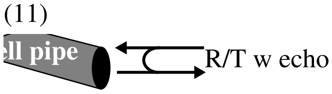
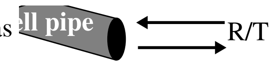
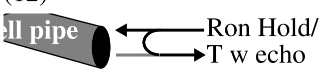
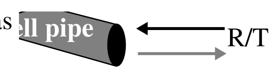
Appendix E: Connection Permissions and Access Mapping Interaction

Table 10 lists the twelve connection mapping types that a client may choose. Certain combinations of offered ep_perm from the network would disallow the indicated mapping and those are listed to the right of the mapping description in the rightmost column in the table. It is sufficient for any one of the conditions to be met for the mapping to be disallowed. For instance, if a client desired a mapping as a *transmitter with echo* and the offered ep_perm had the specification for the Receive side of the connection as <Receive = <perm=ON, changeable=OFF> (which says that all clients in this connection <u>must</u> be receivers unless specifically authorized by the owner), the mapping would be disallowed to the client regardless of the Transmit side specification. Likewise, a Transmit side specification of <Transmit = <perm=OFF, changeable=OFF> (which says all clients may <u>not</u> be transmitters), would disallow the mapping.

TABLE 10. Connection Permissions and Access Mapping Interaction

Client Connection Access	**Endpoint Access Mapping**	**<u>Disabling</u> Endpoint Permissions[1] (on an invite to a client from the network) (i.e. what would disallow the indicated mapping)[2]**
(1) ll pipe Null	Receive = OFF Transmit = OFF Echo = OFF	<Receive = <perm=ON, changeable={OFF}> <Receive = <perm=HOLD, changeable={OFF}> <Transmit = <perm=ON, changeable={OFF}> <Transmit = <perm=HOLD, changeable={OFF}>
(2) ll pipe T	Receive = OFF Transmit = ON Echo = OFF	<Receive = <perm=OFF, changeable={OFF}>[3] <Receive = <perm=HOLD, changeable={OFF}>[4] <Transmit = <perm=OFF, changeable={OFF}> <Transmit = <perm=HOLD, changeable={OFF}>
(3) ll pipe T on Hold	Receive = OFF Transmit = HOLD Echo = OFF	<Receive = <perm=OFF, changeable={OFF}> <Receive = <perm=HOLD, changeable={OFF}> <Transmit = <perm=OFF, changeable={OFF}>
(4) ll pipe T w echo	Receive = OFF Transmit = ON Echo = ON	same as ll pipe T
(5) ll pipe R only	Receive = ON Transmit = OFF Echo = OFF	<Receive = <perm=OFF, changeable={OFF}> <Receive = <perm=HOLD, changeable={OFF}> <Transmit = <perm=ON, changeable={OFF}> <Transmit = <perm=HOLD, changeable={OFF}>
(6) ll pipe R only	Receive = HOLD Transmit = OFF Echo = OFF	<Receive = <perm=OFF, changeable={OFF}> <Transmit = <perm=ON, changeable={OFF}> <Transmit = <perm=HOLD, changeable={OFF}>
(7) ll pipe R/T	Receive = ON Transmit = ON Echo = OFF	<Receive = <perm=OFF, changeable={OFF}> <Receive = <perm=HOLD, changeable={OFF}> <Transmit = <perm=OFF, changeable={OFF}> <Transmit = <perm=HOLD, changeable={OFF}>
(8) ll pipe R/T	Receive = ON Transmit = HOLD Echo = OFF	<Receive = <perm=OFF, changeable={OFF}> <Receive = <perm=HOLD, changeable={OFF}> <Transmit = <perm=OFF, changeable={OFF}>

TABLE 10. Connection Permissions and Access Mapping Interaction (Continued)

Client Connection Access	Endpoint Access Mapping	Disabling Endpoint Permissions[1] (on an invite to a client from the network) (i.e. what would disallow the indicated mapping)[2]
(9) 	Receive = HOLD Transmit = ON Echo = OFF	<Receive = <perm=OFF, changeable={OFF}> <Transmit = <perm=OFF, changeable={OFF}> <Transmit = <perm=HOLD, changeable={OFF}>
(10) 	Receive = HOLD Transmit = HOLD Echo = OFF	<Receive = <perm=OFF, changeable={OFF}> <Transmit = <perm=OFF, changeable={OFF}>
(11) 	Receive = ON Transmit = ON Echo = ON	same as 
(12) 	Receive = HOLD Transmit = ON Echo = ON	same as 

[1] The owner may always override default permissions if it invites the client.

[2] In the table, it is sufficient for only one disabling permission (receive or transmit) to disallow the indicated access mapping.

[3] Combined permissions of this form, disallowing both transmitters and receivers as a default seem nonsensical. Since the owner may override defaults, however, it may be advantageous.

[4] The HOLD is different than the OFF option, it keeps the bandwidth free at the client's termination link but held throughout the network, which provides an reduced blocking artifact if **mod_ep** changes are frequent..