Connection Management Network Protocol (CMNP) Specification

John DeHart Dakang Wu

Version 1.0 (Still incomplete)
July 3, 1996

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

ABSTRACT

This document specifies a *Connection Management Network Protocol* (CMNP) for managing multipoint multimedia communications in high-speed packet switched networks. We target CMNP 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. CMNP specifies message formats used to pass the control information among network nodes to create, modify and delete multipoint connections. CMNP also defines the actions at network nodes when a message is received. Once a connection is established, network nodes exchange data using protocols other than CMNP.

1. Introduction

This document describes a communication architecture and an information model for multipoint connections in a switched Asynchronous Transfer Mode (ATM) network and specifies the Connection Management Network Protocol (CMNP) that allows network nodes to work together to create, manipulate and delete multipoint, multiconnection communication channels. These multiconnection communication channels are called a *connection group* or a *call* from user's point of view. A *call* is a concept at the user level or at UNI, whereas a *connection group* is a concept at the network level or NNI. 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 an exterior network node and the client who created the call. This client is designated the *owner* of the *call*. Additional clients, or *endpoints*, can be 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.

Network nodes have to pass messages in order to realize the external requests such as call creation, call maintenance and connection maintenance. CMNP defines the interface between network nodes to create, manipulate and delete connection groups and connections within a connection group. As such, CMNP is an ATM *Network Node Interface* (NNI) signaling protocol [3] [14]. It is layered over a reliable substrate, which we term CTL (CMNP 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). In this way, CMNP implementors can choose the most suitable CTL for their implementation environment.

Addressing is another area where CMNP remains flexible by not dictating any one addressing scheme. Rather, CMNP supports multiple addressing disciplines and multiple routing protocols. Currently defined addresses schemes include IP addressing [18], public network E.164 addressing [13], and OSI NSAP addressing [17]. An implementation of CMNP 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**	CMNP overview.
Section 2: **ATM Networks**	Introduction to switched, connection oriented ATM networks.
Section 3: **CMNP Concepts**	Concepts and Information Model
Section 4: **Connection Group Model**	A Connection Group model for CMNP
Section 5: **CM Access Protocol**	Discussion of the relations between access protocols and CMNP
Section 6: **CMNP Operations**	Message formats and state diagrams of all CMNP operations.
Section 7: **Future Directions**	List of enhancements being considered.
Appendix A: **References**	List of related references.
Appendix B: **CMNP Field Values**	Numerical values for CMNP message parameters.

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. The nodes (exterior and interior), under the control of a network protocol, like CMNP, exchange control messages, in order to satisfy the request. The access protocol hides the internal topology of the network from clients.

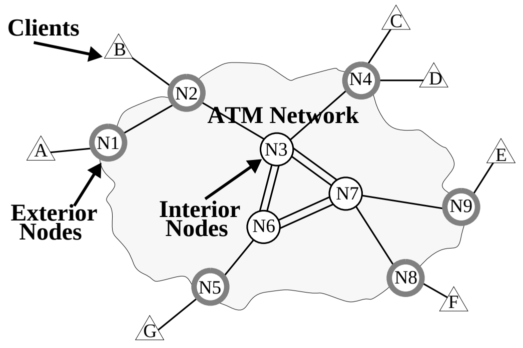


Figure 1. Example ATM Network.

Each node in the network contains one or more ATM switches [29][33][34][36][37][58][61]. The switches route each ATM cell 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 a connection setup period. Connectionless traffic can be accommodated via *overlay* networks utilizing special purpose routers or datagram processors [5][9].

Figure 2 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. Multicast cells, destined for several locations, are replicated by the CN, then routed to the appropriate destination by

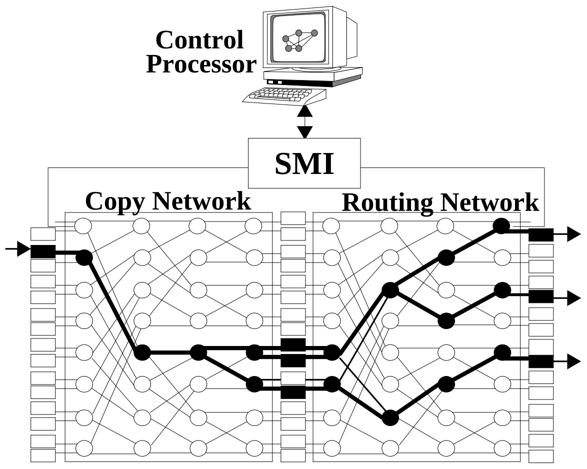


Figure 2. Architecture of Turner's Broadcast Packet Switch Demonstrating Multicast Routing.

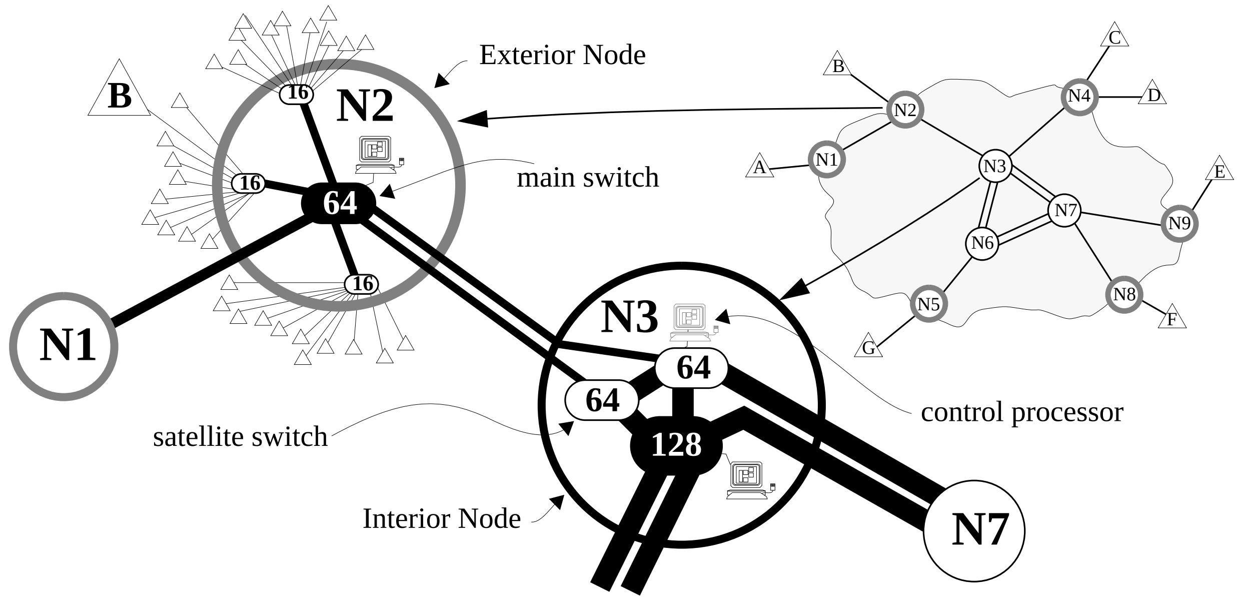


Figure 3. Generalized Node Architecture for Interior and Exterior Network Nodes.

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.

{Add GBN Switch Description}

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][ATM UNI Ref] 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 NNI cell format is shown in Figure 4. It consists of a 48 byte information (data) field and a five byte header. The header has five fields: a Virtual Path Identifier (VPI, 12 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).

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, the network uses the VPI for routing, remapping this field at every switching node within the network, until the cells reach their destinations. With VC connections, 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 connections 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 nodes receiving a cell that congestion was encountered

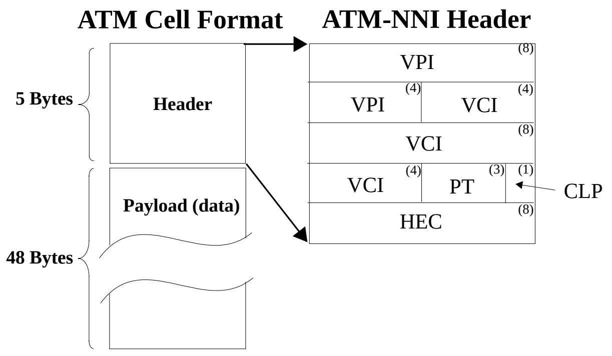


Figure 4. ATM UNI Cell Format.

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 as in AAL5 [9][42][AAL5 reference]. 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

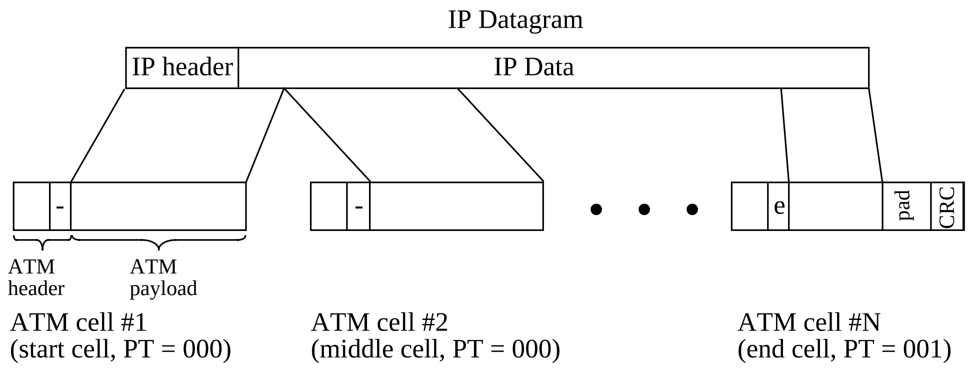


Figure 5. IP Segmentation/Reassembly Scheme.

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. 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.

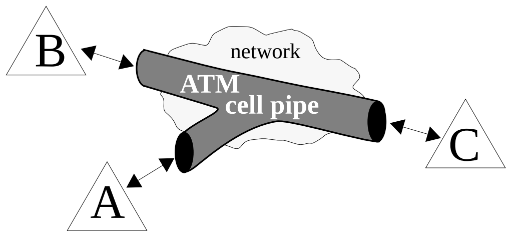


Figure 6. Three-way Cell Pipe.

2.4 Virtual Path and Virtual Channel Connections

Clients can 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 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).

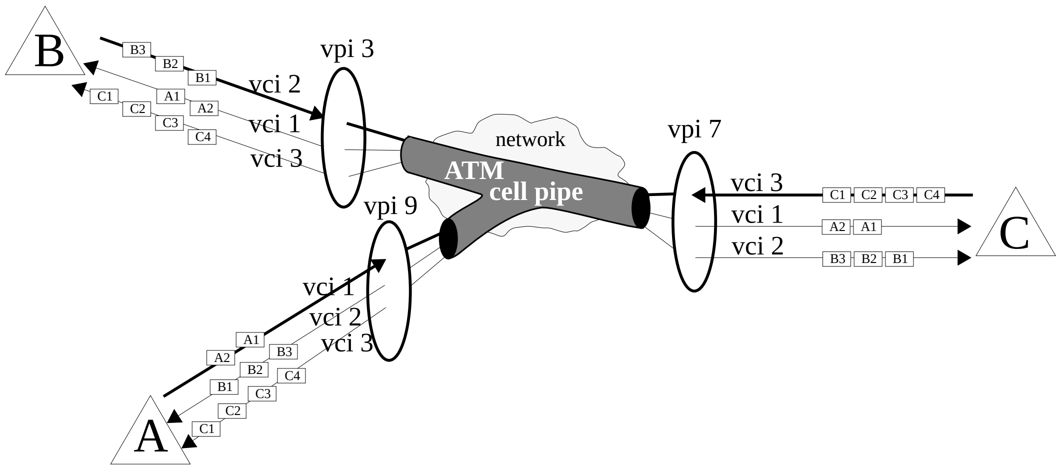


Figure 7. Sequenced Bidirectional Multipoint VP Connection Using the VCI for Source Discrimination.

2.5 VPI/VCI Assignment at the NNI

Our model supports both *uni-directional* and *bi-directional* assignment of VPIs and VCIs at the NNI. 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, one node allocates a VPI/VCI table entry and informs the adjacent node that it is to use the assigned VPI/VCI when sending cells to it. This resource management information is sent with a request. The adjacent node indicates the VPI/VCI for the opposite direction when it acknowledges the operation. Each node allocates the resources according to this algorithm, obviating the need for VPI/VCI assignment collision detection, avoidance and recovery.

At the NNI, we allow a node 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. A node can also leave both pairs empty, in which case the partner will assign both (choosing identical values, if a strictly bi-directional assignment policy is chosen by the CMNP implementors).

3. CMNP Concepts

There are several concepts on which CMNP based. In section 3.1, we introduce the Communication Architecture, which defines the functionality and the scope of CMNP. In section 3.2 we discuss the problem of object layering, how the objects are defined, organized, and how they communicate each other. Section 3.3 gives a connection-group model. Section 3.4 discusses some implementation issues such as transaction, naming and addressing.

3.1 Communication Architecture

With respect to network control and signaling, some emerging requirements of the telecommunication industry have been identified: (1) support for information networking in various media; (2) support for multipoint communication; (3) support for a diverse set of telecommunication services such as dynamically allocating bandwidth and QOS; (4) rapid introduction of new services; (5) modularity and interoperability of multi-supplier solutions and so on.

A Communications Architecture has been defined to meet these requirements. It includes the following principles:

(1) separation of communication control from switching and transmission resources

- service development and evolution can be independent of vendor-specific technologies and implementation architectures
- switching and transmission resources can be represented using a generic managed object approach

(2) separation of call (connection group) control and connection control

- access signaling and call processing can be independent of connection control
- multiple access signaling protocols can be supported
- connection control can be re-used by different services

The Communications Architecture includes the following elements: the Session Management Functional Area (SMFA), the Connection Management Functional Area (CMFA), Network Resources (NR), and the Customer Premise Network (CPN). Figure 1 shows the relationship among these elements.

The CPN represents end-user's telecommunication equipment. A CPN is modeled as a Session Manager (SM) client. It communicates with the SMFA through the Session Manager Interface (SMI). A single CPN may have one or more SM clients, each uses a separate SMI. End-user information is transmitted and received across the Transport Interface (TI). The TI may support multiple end-user information streams simultaneously, each is referred to as a channel. Generally, each SMI is implemented as a separate signaling channel which shares a User-Network Interface (UNI) with all the channels comprising the TI.

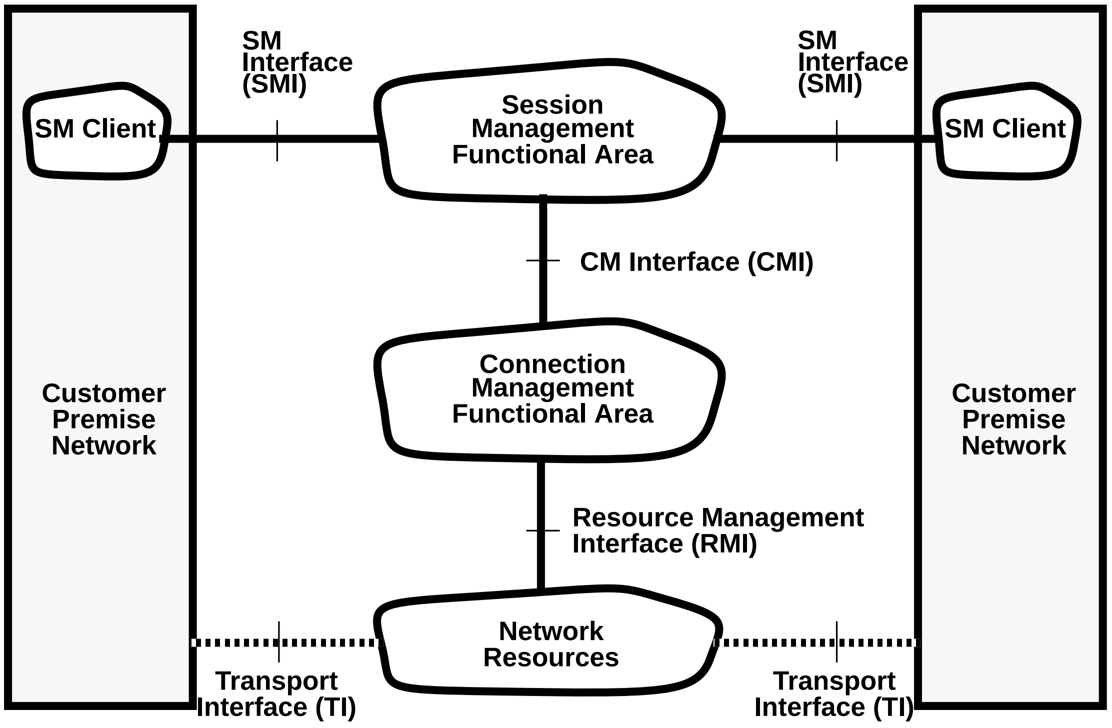
The Session Management Functional Area (SMFA) acts on behalf of end-users, or clients, in support of session (call) establishment, modification, and release. The SMFA interfaces the client at one side through Session Manager

Interface (SMI). On the other side, it interfaces the CMFA through Connection Management Interface (CMI) to request network connectivity operations.

The Connection Management Functional Area (CMFA) performs the task of establishing, modifying, and releasing connections in a telecommunication network. The clients of the CMFA may be the SMFA or administrative applications which manage CMFA. An SMFA interfaces the CMFA through the Connection Management Interface (CMI), whereas the administrative applications interfaces the CMFA through the Connection Management Administration Interface (CMAI). The CMFA interfaces the Network Resources layer through Resource Management Interface (RMI).

Network Resources represent the collection of switches, transmission, and adaptation systems which support the transport and processing of end-user information.

Figure 8. Communication Architecture



3.1.1 Connection Management Functional Area

The CMFA encompasses the complete functionality necessary to control all network resources for purposes of establishing, modifying, and releasing network connections.

A CMFA should have following functionalities. It should: provide a simple yet powerful model of abstract network connectivity to clients of the CMFA; flexibly support connections with a wide range of bandwidths, QOS, and network topologies; select and manage network resources in order to meet a client’s specified connectivity and quality of service requirements; perform routing between network interfaces; provide connectivity control to the equipment that lacks that functionality.

Additionally, the CMFA should have the following characteristics: use industry standards as appropriate; support a distributed implementation.

For practical reasons of processing load, scale, geography, administrative partitioning, flexibility, reliability, etc., the CMFA will be implemented as a distributed collection of discrete connection management functional instances called Connection Managers (CMs). Each CM will exercise exclusive control over a collection of network resources called a Node. Each Node has one and only one CM.

A Node, and hence a CM, has an address which uniquely identifies it and allows other CMs to communicate with it. A CM address format is shown in Figure 9.

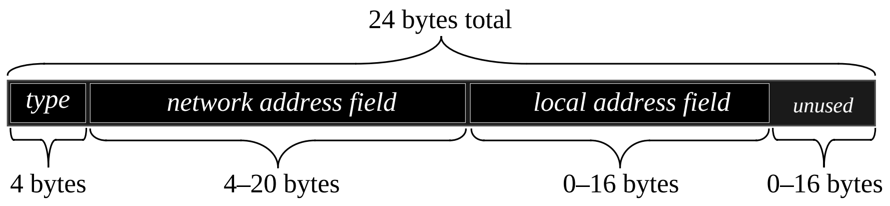


Figure 9. Node Address Format

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 10. The network address field contains the only portion of the address that the network uses in node identification and routing. The local address field is passed end-to-end for use by clients and is not interpreted by the network. The remainder of the address is unused.

The first diagram of Figure 10 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. CMNP 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 10 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 10 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.

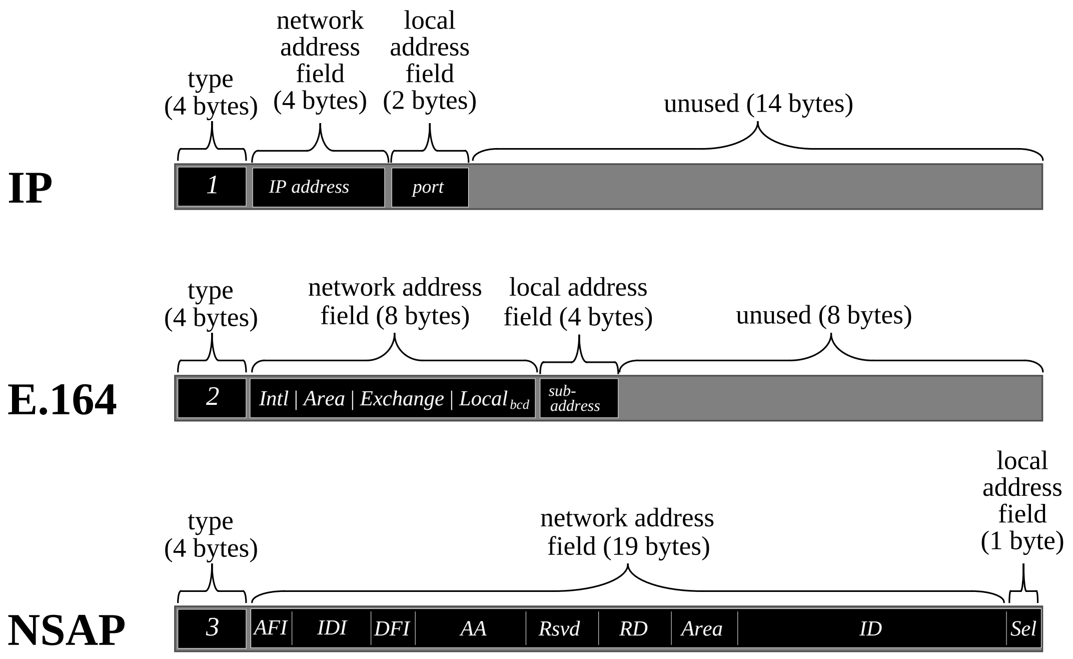

IP
type (4 bytes)
network address field (4 bytes)
local address field (2 bytes)
unused (14 bytes)
1
IP address
port
E.164
type (4 bytes)
network address field (8 bytes)
local address field (4 bytes)
unused (8 bytes)
2
Intl | Area | Exchange | Local bcd
sub-address
NSAP
type (4 bytes)
network address field (19 bytes)
local address field (1 byte)
3
AFI
IDI
DFI
AA
Rsvd
RD
Area
ID
Sel

Figure 10. Examples of Client Address Partitioning.

A CM may have one or more Session Managers attached and/or rooted at it.(See description of SMFA below)

3.1.2 Session Management Functional Area

The SMFA encompasses the complete functionality necessary to manage telecommunication sessions on behalf of SM Clients.

The SMFA should have following functionalities. It should: terminate an access signaling protocol (e.g.,CMAP, Q.93B) and support a call model for clients; negotiate call parameters on the behalf of clients; allow clients to subscribe to and register for various services provided by the SMFA; perform name to logical address translations; publish the names/addresses of clients; perform security checks; use the CMFA to request network connectivity operations.

For practical reasons of processing load, scale, geography, administrative partitioning, flexibility, reliability, etc., the SMFA will be implemented as a distributed collection of discrete session management functional instances called Session Managers (SMs).

An SM is of a certain type such as CMAP, Q.93B, etc. SMs may communicate with one another in order to negotiate on behalf of their respective clients. The SM's network address is based on the node address and a local iden-

Node_Address	**Local_SM_id**

tifier that uniquely identifies it at that node.

A Session Manager uses the CM to which it is attached to manipulate connection groups by utilizing operations provided by the CM to create, modify and destroy connection groups.

3.2 Information Model

Information models are convenient for defining the interface between two systems which must communicate (e.g., peer-to-peer, client-server, managing-managed). Communicating systems exchange information across their interface, and an information model allows the two systems to view that information identically. While there are several different information modeling methodologies and tools, we advocate the approach of using “managed objects”.

The clients of CM (e.g., Session Managers) which request its services to manipulate connections access it at a service interface called CM Interface (CMI) and communicate their requests in terms of an operational view of the CM information model. In a similar manner, administrative functions which manage CM access it at management interface called the CM administrative interface (CMAI) and use a management view of the information model when performing management operations on CM. In this section the objects comprising the CM information model are introduced, and their relationships are described.

3.2.1 Objects

3.2.1.1 Subnetwork

A subnetwork object represents some collection of network transport resources (e.g., switching and transmission systems). It defines the domain of a CM. The CM controls the resources represented by the subnetwork for purposes of establishing, modifying, and releasing connections.

Connections in a subnetwork are established between external interfaces to the subnetwork. These interfaces, or LTPs, are defined below.

The subnetwork object is static, in the sense that it is instantiated at configuration time.

3.2.1.2 Link Termination Point (LTP)

An LTP is an object which represents the interface to a subnetwork. An LTP is contained by the subnetwork object to which it represents an interface. An LTP serves as a container for a collection of Connection Points (CPs) which are located at the same topological location. In an ATM VC or VP layer network an LTP is an object that represents an ATM UNI or NNI.

LTPs are static. They are instantiated by configuration management at network initialization.

3.2.1.3 Connection Point (CP)

A CP is an object which represents the origination or termination of information flow (from the subnetwork’s perspective). A CP is either a transmitter or a receiver of information. The bandwidth attribute of a CP may be modified by a CM but not directly by a client of CM. In an ATM VC layer network a CP represents a virtual channel.

CPs are logically static, as far as the clients of CM are concerned. They are instantiated by configuration management at network initialization. Their creation and destruction, if they occur, are transparent to the clients of CM.

3.2.1.4 Connection Group (CG)

A connection group is an object that contains several connections. A CG has a network wide unique identifier consisting of an SM address and a local connection group id which uniquely defines the CG within the SM.

SM_Address	**Local_CG_id**

The CG identifier is specified by SM at creation and is verified for uniqueness by CM. (For our implementation of CMNP, the Local_CG_id will likely be, the local call id LCID from a UNI concatenated with the SM type)

CGs are dynamic. Clients of CM can request the creation, modification, or destruction of CGs.

3.2.1.5 Connection (or Subnetwork Connection)

A connection is an object that represents the association of zero or more CPs (via contained edges) for the purpose of transferring information across a subnetwork.

Receiver CPs receive the merged information streams of all transmitter CPs in the connection. In a VC layer net, the information streams are merged onto a single channel. In a VP layer net, the information streams are merged onto a single VP; VCIs retain end-to-end significance.

~~If a receiver CP and a transmitter CP in the same LTP point to the same edge object (see below), then the receiver CP can optionally receive or not receive the information stream of its paired transmitter CP. This is referred to as having echo on or off, respectively.~~

Connections are dynamic. Clients of CM can request the creation, modification, or destruction of connections.

3.2.1.6 Edge

An edge is an object that represents the association of CP(s) into a connection. An edge may associate a single transmitter CP, a single receiver CP, or a transmitter and receiver CP pair into a connection.

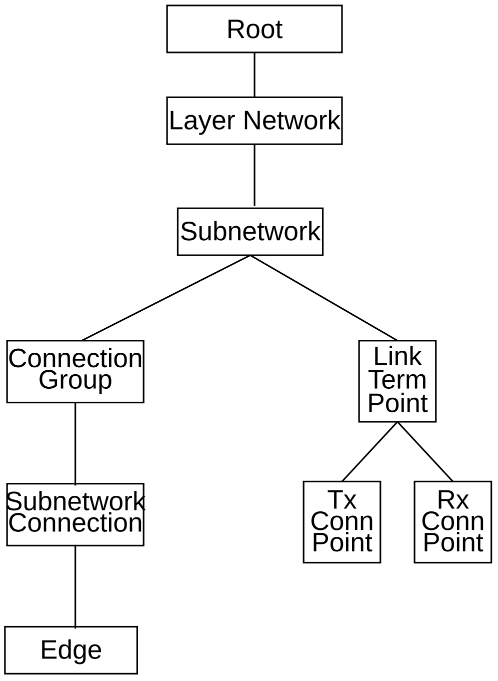
Edges have a directionality attribute representing the direction of the information flow: Transmit (Tx) represents the information flowing into the connection from a Tx CP; Receive (Rx) represents the information flowing out of the connection from a Rx CP; Transmit/Receive (Tx/Rx) represents the information flowing into and out of the connection from a pair of Tx and Rx CPs contained in the same LTP. ~~If the directionality attribute is set to Tx/Rx, the edge echo attribute becomes meaningful and represents whether the received information stream contains a copy of the transmitted information stream.~~

Edges are dynamic. Clients of CM can request the creation, modification, or destruction of edges.

3.2.2 Containment Hierarchy

Figure 11 shows the containment hierarchy of the objects.

Figure 11. Containment Hierarchy



3.3 Some Implementation Issues

In this section, we discuss some implementation issues related to the CMNP.

3.3.1 Transaction

A CM provides a capability to SMs for grouping multiple related operations into a transaction. A transaction may have its state altered as an atomic unit. This gives an SM the ability to make multiple changes and have them all take effect at once or none at all. In reality, of course, all changes cannot be performed simultaneously but with transactions the CM can reserve all the operations in one transaction, and then commit them all together.

For example to build a connection that has one transmitter and four receivers, an SM can add all the participants in one transaction and then commit the transaction. This will allow the CM to do the same with the Switch Side Managed Object (SSMO) level. When the CM commits the transaction in the SSMO, the hardware will be updated for one broadcast of one to four instead of: first setting up a point to point, then a one to two, then a one to three, and then finally a one to four.

The reserve and commit operations are asynchronous, meaning that their responses will not be immediate. The reserve operation will check the validity of the reserve operation. If the operation is not valid, an error indication will be returned. Otherwise a no-error indication will be returned. Later, the CM will provide to the requester of the operation an asynchronous response, reporting the result of the operation.

When an SM creates a transaction, its CM returns to it a transaction id. This id is used to identify to which transaction each operation belongs. When performing CMNP operations to accomplish an operation in a transaction, the CM will include the transaction id in the CMNP operation PDU. Multiple transactions may exist at the same time.

An SM may pass a transaction id to other SMs in order to coordinate network wide operations. (e.g. have all edges added to a connection in one transaction.)

3.3.2 Rendezous

CMs have to synchronize their operations. For example, when a NET_JOIN_CG message is sent, the sender has to wait until a response comes back. Rendezous provides a means to synchronize among CM operations. When an operation has to synchronize with other operations, it requests a rendezous mark, which is unique in the network system. The rendezous marks are sent with the message which has to be responsed. When a response comes back, the rendezous mark guides the system to find which operation can resume.

Rendezous is not the only way to synchronize in a multiprocess system, but it is a convenient way to implement the CMNP.

4. Connection Group Model

Connection-Groups (CG) are the basic concept of communication linkage at the network node level. From the end-user's point of view, a CG presents as a call. A connection group is a distributed object maintained by network nodes that describes the communication paths and their attributes. The attributes of CGs and the operations that can be performed on them define the *connection group model*.

A CG contains a set of communication channels, which we term connections. Initially, at a CG creation, a connection group has one or more connections. The SM that initiates the CG creation is assigned as the owner of that CG. The owner of a **CG** (and other SMs with the owner's permission) can add connections to or remove connections from the CG. Multiple connections within a CG are useful for applications such as video conferences, where one connection may carry video and another audio. CGs are dynamic, in the sense that the number of participates and the number of connections in the CG can vary over time, while communication is taking place. Additionally, the bandwidth reserved by a connection can also be modified.

4.1 Connection Group (CG)

Connection Groups (CG) are the primary objects manipulated by CMs. Figure 12 shows an example of a multimedia CG at a Node. CGs have a number of attributes that are described in the following subsections and summarized in Table 3.

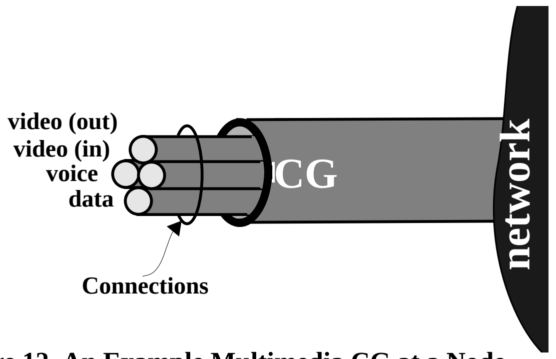


Figure 12. An Example Multimedia CG at a Node.

4.1.1 CG_Owner

The CG_Owner attribute is an SM identifier indicating who initially created the CG. The SM identifier format has been defined in section 3.1.2.

TABLE 3. Connection Group attributes

attribute	**Description**
CG_Owner	the SM id who creates the CG
CG_Ack_Flag	indicating whether the owner of the CG has to approve before a new end-point can be added into the CG
CG_ID	Connection Group identifier
CG_State	the CM state regarding to the CG
CG_Correlation	whether all the connections in the CG have to be routed on the same path
CG_Trunking	use trunk whenever possible
Connection_List	Current connections in call

4.1.2 CG_Ack_Flag

The CG_Ack_Flag is a numerate variable that indicates who has the right to add a new end-point into a CG.

$$\text{CG\_Ack\_Flag} \in \{\text{ON, OFF, CLOSED}\}$$

When CG_Ack_Flag is ON, the CG owner has to approve the addition of a new end-point. When CG_Ack_Flag is OFF, end-points can be added freely. When CG_Ack_Flag is CLOSED, only the owner of the CG can request to add a new end-point into the CG.

4.1.3 CG_ID

The CG_ID is the connection group’s identifier which is globally unique. The CG_ID format is defined in section 3.2.1.4.

4.1.4 CG_State

The CG_State attribute gives the CG’s current state at the node.

$$\text{CG\_State} \in \{\text{REQUEST, RESERVE\_PENDING, RESERVE, COMMIT\_PENDING, COMMIT, ABORT\_PENDING, ABORT}\}$$

4.1.5 CG_Correlation

The correlation attribute controls the way to choose a link for a connection:

$$\text{CG\_Correlation} \in \{\text{YES, NO}\}$$

If the correlation is YES, all the connections within the CG have to be routed on the same path within a subnetwork. If the correlation is NO, the connections can be routed separately.

4.1.6 CG_Trunking

The trunking attribute tells if a predefined trunk to be used to route the connections if possible.

$$\text{CG\_Trunking} \in \{\text{YES, NO}\}$$

4.1.7 Connection_List

The Connection_List is a list of connection objects within the CG. Each connection includes allocated bandwidth, connection type, and other attributes as described in the next section. The list is dynamic since connections can be added to or removed from the connection group at any time, and since the attributes of connections can be changed over time.

4.2 Connections

Connections are the primary information carrying components of a connection group. The attributes associated with a connection are described in the following subsections and summarized in Table 5. Some of connection attributes have global meaning. Some have local meaning since they are different at different nodes.

TABLE 4. Connection attributes.

attribute	**Description**
Con_Owner	a SM identifier who created this connection
Con_Ack_Flag	indicating whether the owner of the connection has to approve before a new end-point can be added into the connection
Con_ID	connection identifier which is unique within a CG
Con_State	connection state
Con_Type	a three tuple consisting of VP/VC, dynamic/static bandwidth and quality of service
rx_bw	receive bandwidth requirement
tx_bw	transmit bandwidth requirement
QOS	quality of service
Con_Edge_list	a list of edges

4.2.1 Con_Owner

The Con_Owner attribute is an SM identifier indicating who initially created the connection. The SM identifier format has been defined in section 3.1.2.

4.2.2 Con_Ack_Flag

The Con_Ack_Flag is a numerate variable that indicates who has the right to add a new end-point into a connection.

$$\text{Con\_Ack\_Flag} \in \{\text{ON, OFF, CLOSED}\}$$

When Con_Ack_Flag is ON, the connection owner has to approve the addition of a new end-point. When Con_-Ack_Flag is OFF, end-points can be added freely. When Con_Ack_Flag is CLOSED, only the owner of the connection can request to add a new end-point into the connection.

4.2.3 Con_ID

The Con_ID is an integer which uniquely identifies the connection within the CG.

4.2.4 Con_State

The Con_State attribute gives the connection's current state at the node.

$$\text{Con\_State} \in \{\text{REQUEST, RESERVE\_PENDING, RESERVE, COMMIT\_PENDING, COMMIT, ABORT\_PENDING, ABORT}\}$$

4.2.5 Con_Type

The con_type attribute gives the type of the connection.

$$\textbf{Con\_Type} \in \{\textbf{VP}, \textbf{VC}\}$$

4.2.6 rx_bw

The rx_bw attribute is a three tuple that defines the receive bandwidth.

$$\text{rx\_bw} = < \textbf{peak}, \textbf{average}, \textbf{peak\_burst\_length} >$$

The *peak* and *average* attributes are expressed in cells per second. The *peak burst length* is measured in cells and indicates the maximum number of cells that the connection can receive at peak rate during a burst.

4.2.7 tx_bw

The tx_bw is a three tuple that defines the transmit bandwidth in the same format as rx_bw.

4.2.8 QOS

The QOS attribute defines the quality of service.

$$\text{QOS} \in \{\text{HIGH, MEDIUM, LOW}\}$$

QOS relates to options for cell loss behavior that may vary from network to network. QOS, therefore, is intentionally vague. If the network can implement different loss behavior strategies the network control software will group them into the categories of HIGH, MEDIUM and LOW.

4.2.9 Con_Edge_List

The Con_Edge_List is a list of edge objects. Each edge object defines an input or output port of the connection. Edge objects are described in detail in next section.

4.3 Edges

An edge is an object that represents the association of CP(s) into a connection. An edge may associate a single transmitter CP, a single receiver CP, or a transmitter and receiver CP pair into a connection. The attributes associated with an edge are described in the following subsections and summarized in Table 5.

TABLE 5. Edge attributes.

attribute	Description
Eg_Owner	a SM identifier identical to Con_Owner
Eg_Direction	indicating the direction of the edge: in, out, or bidirectional
Eg_echo	receive messages of its own or not
Eg_State	edge's current state
rx_bw	receive bandwidth requirement

TABLE 5. Edge attributes.

attribute	Description
Eg_Owner	a SM identifier identical to Con_Owner
Eg_Direction	indicating the direction of the edge: in, out, or bidirectional
Eg_echo	receive messages of its own or not
Eg_State	edge's current state
tx_bw	transmit bandwidth requirement
Rx_CP	receive connection point
Tx_CP	transmit connection point

4.3.1 EG_Owner

The Eg_Owner attribute is an SM identifier indicating who initially created the edge. This is same as the owner of the connection. The SM identifier format has been defined in section 3.1.2.

4.3.2 Eg_Direction

The Eg_Direction indicates the direction of the edge.

$$\text{Eg\_Direction} \in \{\text{RECEIVE, TRANSMIT, RXTX}\}$$

The direction of an edge can be RECEIVE, TRANSMIT, or both receive and transmit that means bidirectional.

4.3.3 Eg_echo

Eg_echo is a boolean variable indicating whether echo is allowed. This attribute is valid only when the edge is a TRANSMIT edge or a RXTX edge.

4.3.4 Eg_State

The Eg_State attribute gives the edge's current state.

$$\text{Eg\_State} \in \{\text{REQUEST, RESERVE\_PENDING, RESERVE, COMMIT\_PENDING, COMMIT, ABORT\_PENDING, ABORT}\}$$

4.3.5 rx_bw

The rx_bw attribute is a three tuple that defines the receive bandwidth. The format of rx_bw is same as the format of rx_bw in the connection object (section 4.2.5). The difference is the bandwidth here defines the actual bandwidth on a specific link, and the bandwidth in the connection object defines the bandwidth requirement which may or may not exist on the current edge.

4.3.6 tx_bw

The tx_bw is a three tuple that defines the transmit bandwidth in the same format as rx_bw.

4.3.7 Rx_CP and Tx_CP

The Rx_CP and Tx_CP are the receive-connection-point and transmit-connection-point objects respectively. By containing these objects, the edge object, which is dynamic, sets up a relationship between the connection and the connection points, which are logically static and are related to the resources managed by the node. Connection point object is described in next section.

4.4 Connection Point

A CP is an object which represents the origination or termination of information flow (from the subnetwork's perspective). A CP is either a transmitter or a receiver of information. A transmit-CP is an information source from which the subnetwork receives cells. A receive-CP is an information sink to which the subnetwork sends cells. In each type, we distinguish VP-CP from VC-CP which represent virtual-path CP and virtual-circuit CP respectively. The attributes associated with a CP are described in the following subsections and summarized in Table 6.

TABLE 6. Connection-Point attributes.

attribute	**Description**
CP_address	the link address the CP resides in
CP_Type	distinguishes TXCP from RXCP, VC from VP
CP_I	either VCI of VPI decided by CP_Type

4.4.1 CP_address

The CP_address attribute gives the link address in which the CP resides. The address format is an implementation decision.

4.4.2 CP_Type

The CP_Type attribute defines the type of the CP.

$$\text{CP\_Type} \in \{\text{TXVC}, \text{TXVP}, \text{RXVC}, \text{RXVP}\}$$

A CP has one of the four types: TXVC stands for transmit-virtual-circuit-CP, TXVP for transmit-virtual-path-CP, RXVC for receive-virtual-circuit-CP, and RXVP for receive-virtual-path-CP.

4.4.3 CP_I

CP_I is an integer that defines the VCI or VPI depends on the CP_Type.

5. Connection Management Access Protocol

Clients of BISDN networks create, manipulate and destroy calls by sending messages to exterior nodes of the network. These 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. Figure 13 shows a layered view of the protocol architecture.

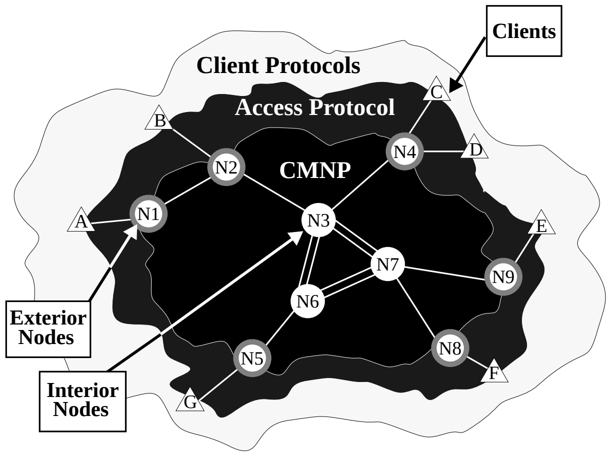


Figure 13. Layered View of the Protocol Architecture.

Clients have no knowledge or visibility of CMNP. CMNP does not restrict the access protocols used at client side. It can even support different access protocols in an ATM network simultaneously. For example some clients may use Q.93B as the access protocol, others may use CMAP. The only restriction is that within a connection group, all the clients have to use same access protocol.

6. CMNP Message Formats and Operations

This section describes the detailed CMNP message formats and the operations at each node when a message is received. Section 6.1 outlines the conventions used to diagram CMNP messages. Section 6.2 describes the common terms, fields and attributes used in CMNP messages. Finally, Section 6.3 gives the message layouts and detailed operation descriptions.

6.1 Message Layout Conventions

Figure 32 shows a diagram of the **NET_JOIN_CG** request message.

IE Type	Data Size (bytes)	Data	Mandatory or Optional	Default Value
OP_TYPE	1	NET_JOIN_CG REQUEST	M	
MULTIPHASE	1	REQUEST	M	
MSG_ID	2	1	M	
TRANS_ID	28	NODE-2, 1	M	
SM_Requester	28		M	
CG_Owner	28		M	
CG_ID	28	NODE-1, 17	M	
Send_REND	28		M	
Receive_REND	28		M	
CG_ACK_FLAG	1		O	OFF
CORRELATION	1	STRONG	O	STRONG
TRUNKING	1	YES	O	NO
CON_ID	1	1	M	
CON_OWNER	28	SM-1	M	
TxBW	12	(5M, 5M, 1)	M	
RxBW	12	(5M, 5M, 1)	M	
ACK_FLAG	1	CLOSED	O	OFF
CON_TYPE	1	VC	O	VC
QOS	1	HIGH	O	HIGH
MAPPING	1	RT		
NOTIFY	28	SM-2		
TxVPI	1	1		
TxVCI	2	21		
RxVPI	1	1		
RxVCI	2	22		
CON_ID	1	2		
CON_OWNER	28	SM-1		
TxBW	12	(30M, 30M, 1)		
RxBW	12	(30M, 30M, 1)		
ACK_FLAG	1	CLOSED		
CON_TYPE	1	VC		
QOS	1	HIGH		
MAPPING	1	RT		
NOTIFY	28	SM-2		
TxVPI	1	1		
TxVCI	2	23		
RxVPI	1	1		
RxVCI	2	24		

The example demonstrates how CMNP messages are presented in this document and the conventions used. A CMNP message is organized as a list of Information Elements (IEs). Each IE contains three fields: an IE code, a data size field, and the data of the IE. Some IEs are mandatory, others are optional. An optional IE may have a default value. In Figure 32, the first three columns are the contents of IEs, the fourth column indicate whether this IE is mandatory or optional, the fifth column gives the default value if the IE is optional.

Related parameters within CMNP messages are grouped into *objects*. The NET_JOIN_CG REQUEST shown in Figure 32 contains all five of the different types of objects (some of which appear multiple times and some are components of a larger object). The five objects are: 1) the Header object, 2) the Rendezvous object, which resides in the header object, 3) the CG Attribute object, 4) the Connection Attribute object, and 5) the Edge Attribute object, which is a subobject of a connection object. IEs in a particular object can appear in random order within that object. The exceptions are the OP_TYPE has to be the first IE of any message, and the CON_ID has to be the first IE in a connection object.

The Header object is made up of the *operation type (op_type), multiphase, message identifier (msg_id), transaction_identifier (trans_id), session manager identifier (SM_requester), connection group identifier (cg_id),* *connection group owner (cg_owner)* and a rendezvous object. In the example the *op_type* and *multiphase* fields are filled in with the actual bit values for the **net_join_cg** and REQUEST respectively.

A Rendezvous object is a component object contained in a header object. It is made up of *sender rendezvous (Send_REND)*, and *receiver rendezvous (Receiver_REND)*. Rendezous is a synchronization mechanism between asynchronous processes. The rendezous used here is an implementation method rather than a part of the protocol.

The CG Attribute object is made up of the *correlation*, *trunking fields*, and *connection group acknowledge flag (cg_ack_flag)*.

The Connection Attribute object is made up of the *connection identifier(con_id), connection owner (con_owner), connection type (con_type), transmitter bandwidth (tx_bw), receiver bandwidth (rx_bw), acknowledegement flag (ack_flag), quality of service (qos), transmit and receive mapping (mapping), notify*, and an edge-attribute object.

An Edge Attribute object is made up of *transmitter vpi (tx_vpi), receiver vpi (rx_vpi), transmitor vci (tx_vci),* and *receiver vci (rx_vci),*

6.2 Common CMNP Terms, Fields and Parameters

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

6.2.1 operation type (*op_type*)

The ***op_type*** is a one byte integer field indicating the message type. CMNP defines 20 message types as shown in table 7.

TABLE 7. CMNP Message Types.

Message Type	**Description**
NET_CREATE_CG	from a SM to a CM to request to create a new CG
NET_JOIN_CG	from a SM to a CM, or from one CM to another to request to join an existing CG
NET_DROP_CG	from a SM to a CM, or from a CM to another to request to drop from a CG
NET_MOD_CG	from the owner of a CG to all the CMs participated in the CG to request to modify the attributes of the CG
NET_DESTROY_CG	from the owner of a CG to all the CMs participated in the CG to request to destroy the CG
NET_QUERY_CG_OWNER	query the owner of a CG about a CM being added to or deleted from the CG
NET_NOTIFY_CG_LIST	from a CM to others or from a CM to a SM to notify the change of a CG
NET_GET_CG	from a CM to another to request the information about a CG
NET_ADD_CON	from a SM to a CM, or from one CM to some others to request to add a connection
NET_JOIN_CON	from a SM to a CM, or from one CM to another to request to join an existing connection
NET_DROP_CON	from a SM to a CM to request to drop from a connection
NET_MOD_CON	from a SM to a CM, or from one CM to some others to request to modify the attributes of a connection
NET_DESTROY_CON	from the owner of a connection to all the CMs participated in a CG to request to destroy the connection
NET_QUERY_CON_OWNER	query the owner of a connection about an edge being added to deleted from the connection
NET_NOTIFY_CON_LIST	from a CM to others or from a CM to a SM to notify the change of a connection

TABLE 7. CMNP Message Types.

Message Type	Description
NET_COMMIT_TRANS	from a SM to a CM, or from one CM to some others to request to commit a transaction
NET_ABORT_TRANS	from a SM to a CM, or from one CM to some others to request to abort a transaction
NET_RESET	inform the neighbors that the current CM has been reset
NET_PING	bounce a message to a neighbor node to make sure it is still alive
NET_STATUS	from a SM to a CM, or from a CM to another to request the status information

6.2.2 multiphase

The ***multiphase*** is a one byte long field indicating the phase within an operation. This IE will have one of the two values: REQUEST or RESPONSE. Basically, the message format for a REQUEST message is different from the message format for the same type of RESPONSE message. The op_type and multiphase fields together determine the message format.

6.2.3 message identifier (msg_id)

We use the message identifier to distinguish the operations in a transaction. With the msg_id, when a response to an operation is received, we can know exactly which request it matches. The msg_id is a two byte integer that is unique within a transaction.

6.2.4 transaction identifier (trans_id)

A transaction is unique globally within a network. To keep a transaction identifier unique globally, a trans_id is a combination of a node address and a local identifier. The uniqueness of node addresses and the unique selection of the local identifier at a node guarantees the uniqueness globally.Figure 14 shows the format of a transaction_id.

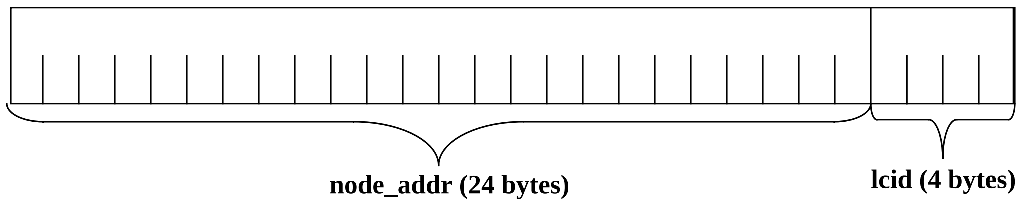


Figure 14. trans_id (28 bytes)

6.2.5 session manager requester (SM_requester)

A session manager requester indicates a unique session manager. See section 3.1.2.

6.2.6 connection group owner (cg_owner)

A CG owner is a SM who creates the CG. The cg_owner is a SM id with the format defined in 3.1.2.

6.2.7 sender rendezvous (Send_REND)

A rendezvous mark is unique in the network system. A concatenation of the node id and a local id which is unique at the node (4 bytes integer) guarantees the global uniqueness.

6.2.8 receiver rendezvous (Receiver_REND)

The format of Receiver_REND is same as Send_REND.

6.2.9 connection group identifier (cg_id)

A CG identifier is unique in a SMFA. The format of cg_id is a concatenation of the node id and a local id which is unique within the SM who creates the CG.

6.2.10 correlation

The correlation parameter takes one of two values: STRONLY or WEAKLY. STRONGLY means all the connections in a CG have to be routed on same path. WEAKLY does not impose that restriction.

6.2.11 trunking

The trunking parameter is a boolean variable. If trunking is YES, each node will try to use a trunk whenever possible.

6.2.12 connection group acknowledge flag (cg_ack_flag)

The cg_ack_flag is a numerate variable that indicates who has the right to add a new end-point into a CG.

$$\text{cg\_ack\_flag} \in \{\text{ON, OFF, CLOSED}\}$$

When cg_ack_flag is ON, the CG owner has to approve the addition of a new end-point. When cg_ack_flag is OFF, end-points can be added freely. When cg_ack_flag is CLOSED, only the owner of the CG can request to add a new end-point into the CG.

6.2.13 connection identifier (con_id)

Each connection is assigned a ***con_id***, unique within a connection group, at the time of the connection's creation. The ***con_id*** is subsequently used to identify which connection is being operated on in any of the connection oriented operations. The con_id is a two byte integer.

6.2.14 connection type (con_type)

Either VC or VP.

6.2.15 transmitter bandwidth (txbw)

Same as defined in 4.2.6.

6.2.16 receiver bandwidth (rxbw)

Same as defined in 4.2.6.

6.2.17 acknowledge flag (ack_flag)

Same as defined in 4.2.2.

6.2.18 quality of service (qos)

Same as defined in 4.2.8.

6.2.19 mapping

The mapping attribute indicates EchoOn or EchoOff at a RxTx edge.

$$mapping \in \{EchoOn, Echo\overline{Off}\}$$

6.2.20 notify

A SM id indicating who will be notified when a new end-node joins athe connection.

6.2.21 transmitter virtual path identifier

Transmitter VPI.

6.2.22 transmitter virtual circuit identifier

Transmitter VCI.

6.2.23 receiver virtual path identifier

Receiver VPI.

6.2.24 receiver virtual circuit identifier

Receiver VCI.

6.2.25 CG_cause

CG_cause and ***Con_cause*** are two IEs contained in a response message to report the execution status of a request.

The ***CG_cause*** field is divided into three subfields. The high order two bits are used to indicate whether there were errors in the CG or connection specifications. If there were errors in the CG specification, the highest order bit is set to 1 (**CG_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 response. If the ***cg_status_bit*** is

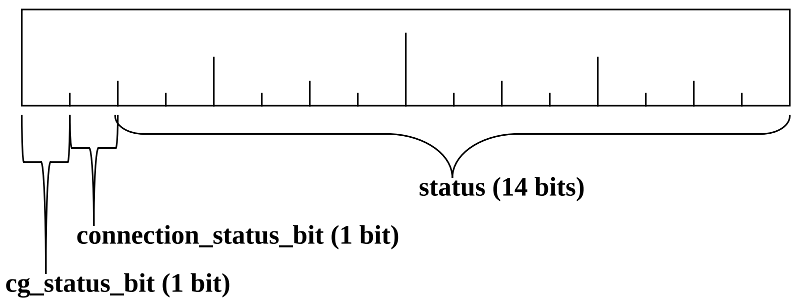


Figure 15. CG_cause (16 bits)

set(***cg_status_bit*** = **CG_SPEC_ERROR**), the client should check the ***status*** subfield (lowest 14 bits of ***CG_cause***) 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_cause*** fields (Section 6.2.26) in the response to determine which connection specification was in error.

In a positive response both error bits are set to 0 (***cg_status_bit*** = **CG_SPEC_OK**, ***connection_status_bit*** = **CON_SPEC_OK**, the rest of the ***CG_cause*** 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.

6.2.26 Con_cause

The ***Con_cause*** 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_cause*** 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.

6.3 CMNP Operation Definitions

Following subsections define the individual CMNP operations. Each operation is defined using three 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. The Operation section describes by prose and state diagrams how the network nodes that received a request should operate.

6.3.1 NET_CREATE_CG

Synopsis:

A NET_CREATE_CG request is a message initiated by a SM. A NET_CREATE_CG response is a message sent by a CM to response to a NET_CREATE_CG request. Having Received a NET_CREATE_CG request, the CM checks its resources. If the resources required are available, the CM will create a new CG, and send an ACK to the SM who initiated the request. Otherwise, a NACK is sent with failure reason in the cause fields. Only when a NET_COMMIT_TRANS request for the transaction in which the NET_CREATE_CG resides is received, the new created CG becomes permanent.

Message Formats:

NET_CREATE_CG request:

IE Type	Data Size (bytes)	Data	Mandatory or Optional	Default Value
OP_TYPE	1	NET_CREATE_CG	M	
MULTIPHASE	1	REQUEST	M	
MSG_ID	2		M	
TRANS_ID	28		M	
SM_Requester	28		M	
CG_Owner	28		M	
CG_ID	28		M	
Send_REND	28		M	
Receive_REND	28		M	
CG_ACK_FLAG	1		O	OFF
CORRELATION	1		O	STRONG
TRUNKING	1		O	YES
CON_ID	1		M	
CON_OWNER	28		M	
TxBW	12		M	
RxBW	12		M	
ACK_FLAG	1		O	OFF
CON_TYPE	1		O	VC
QOS	1		O	HIGH
MAPPING	1			
NOTIFY	28			
TxVPI	1			
TxVCI	2			
RxVPI	1			
RxVCI	2			
CON_ID	1			
		...		

NET_CREATE_CG response:

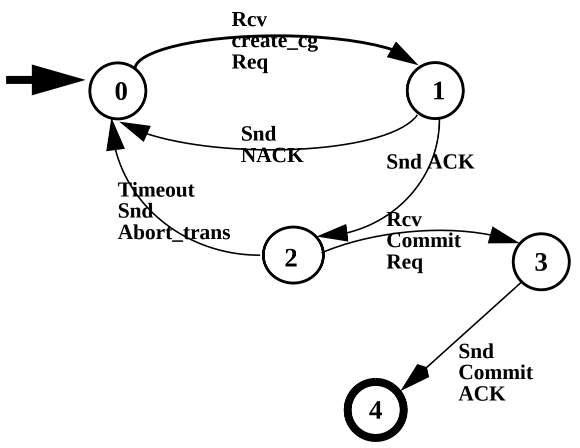
IE Type	Data Size (bytes)	Data	Mandatory or Optional	Default Value
OP_TYPE	1	NET_CREATE_CG	M	
MULTIPHASE	1	RESPONSE	M	
MSG_ID	2		M	
TRANS_ID	28		M	
SM_Requester	28		M	
CG_CAUSE	1		O	OK
CG_Owner	28		M	
CG_ID	28		M	
Send_REND	28		M	
Receive_REND	28		M	
CON_CONTEXT	1		M	
CON_CAUSE	1		O	OK

IE Type	Data Size (bytes)	Data	Mandatory or Optional	Default Value
TxVPI	1			
TxVCI	2			
RxVPI	1			
RxVCI	2			

Operation:

A node starts a net_create_cg operation by receiving a net_create_cg request. Included in the request are a cg_id, a set of connection group parameters, and a set of connection parameters. Figure 16. is the state transition diagram of a CM at a node activated by receiving a net_create_cg request.

Figure 16. net_create_cg state transition diagram



0 -- initial state
1 -- net_create_cg request pending
2 -- waiting for commit request
3 -- commit request processing
4 -- successful net_create_cg

State transitions start at state 0 (no net_create_cg request has been received yet). When a net_create_cg request is received, the node sets up a timer T1, then it goes to the state 1, net_create_cg request pending. At state 1, the node checks its resources to see if the request can be satisfied at the node. If it has enough resources to support the new CG, it will create a temporary CG, reserve the bandwidth on the link where the net_create_cg request came from, and then send an ACK to the requester. Otherwise, a negative net_create_cg response is sent to the requester, and the node goes back to state 0.

When the commit_trans request comes, it goes to state 3, where it cimmits the reservations and sends a positive response back, and then it goes to state 4, the end state for a successful net_create_cg request. At state 2, if the timer T1 expires, the node will release all the resources reserved for the pending connection group, and go back to state 0. In the meanwhile, a net_abort_trans request is sent to the SM to inform the activity.

6.3.2 NET_JOIN_CG

Synopsis:

If the connection group indicated by the ***cg_id*** in the incoming NET_JOIN_CG request exists and the current node has the required resources available, a positive NET_JOIN _CG response is sent to the sender of the request message, which forwards the response back to the origin of the request.

If the connection group has not been created yet, the CM checks its own resources. If the resources required are available, a new connection group will be created temporarily. A new NET_JOIN_CG message, let’s call it the second net_jon_cg request, is sent on the link towards the owner of the connection group. A NET_JOIN_CG response will be sent to the origin of the request based on the success or failure of the second NET_JOIN_CG operation. An initial set of connections within that connection group are established as a part of a successful **net_jon_cg** operation. If the operation fails by receiving a negative response, the reason of the failure will be shown in the cg_cause and con_cause fields in the net_join_cg response.

Message Formats:

NET_JOIN_CG request:

IE Type	Data Size (bytes)	Data	Mandatory or Optional	Default Value
OP_TYPE	1	NET_JOIN_CG	M	
MULTIPHASE	1	REQUEST	M	
MSG_ID	2	1	M	
TRANS_ID	28	NODE-2, 1	M	
SM_Requester	28		M	
CG_Owner	28		M	
CG_ID	28	NODE-1, 17	M	
Send_REND	28		M	
Receive_REND	28		M	
CG_ACK_FLAG	1		O	OFF
CORRELATION	1	STRONG	O	STRONG
TRUNKING	1	YES	O	YES
CON_ID	1	1	M	
CON_OWNER	28	SM-1	M	
TxBW	12	(5M, 5M, 1)	M	
RxBW	12	(5M, 5M, 1)	M	
ACK_FLAG	1	CLOSED	O	OFF
CON_TYPE	1	VC	O	VC
QOS	1	HIGH	O	HIGH
MAPPING	1	RT		
NOTIFY	28	SM-2		
TxVPI	1	1		
TxVCI	2	21		
RxVPI	1	1		
RxVCI	2	22		
CON_ID	1	2		
		...		

NET_JOIN_CG response:

IE Type	Data Size (bytes)	Data	Mandatory or Optional	Default Value
OP_TYPE	1	NET_JOIN_CG	M	
MULTIPHASE	1	RESPONSE	M	
MSG_ID	2	1	M	
TRANS_ID	28	NODE-2, 1	M	
SM_Requester	28		M	
CG_CAUSE	1		O	OK

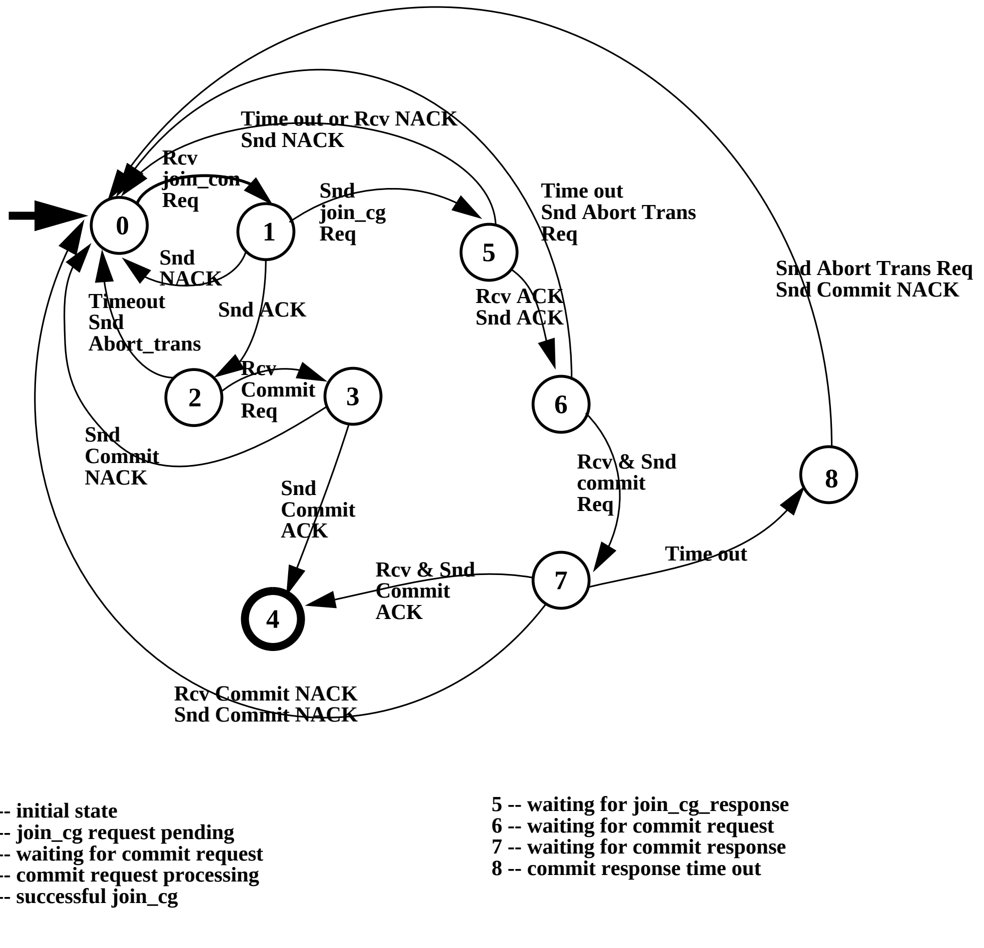
IE Type	Data Size (bytes)	Data	Mandatory or Optional	Default Value
CG_Owner	28		M	
CG_ID	28	NODE-1, 17	M	
Send_REND	28		M	
Receive_REND	28		M	
CON_CONTEXT	1	1	M	
CON_CAUSE	1		O	OK
TxVPI	1	1		
TxVCI	2	21		
RxVPI	1	1		
RxVCI	2	22		

Operation:

A node starts a net_join_cg operation by receiving a net_join_cg request. We refer the sender of the request as requester. Included in the request are a cg_id, a set of connection group parameters, and a set of connection parameters. Based on the cg_ack_flag and ack_flags in connection objects, the state transition diagram can be different.

Figure 17 is the state transition diagram of a CM at a node activated by receiving a NET_JOIN_CG request where all the cg_ack_flag and ack_flags are off.

Figure 17. net_join_cg state transition diagram



State transitions start at state 0, no net_join_cg request has been received yet. When a net_join_cg request is received, the node sets up a timer T1, then it goes to the state 1 (a net_join_cg_request pending). At state 1, the node checks its resources to see if the request can be satisfied at the current node. If there is enough spare bandwidth on the link where the request came from, it reserves the bandwidth on that link. Otherwise, a negative net_join_cg response is sent to the requester, and the node goes back to state 0.

If the connection group indicated by the cg_id in the net_join_cg message has already existed at the node and there are enough resources available at the current node to satisfy the request, the node will send a positive net_join_cg response to the requester, then it goes to state 2 (waiting for a net_commit_trans request). When the net_commit_trans request comes, it goes to state 3, where it commits all the reservations and sends a positive response back, and then it goes to state 4 (the end state of a successful net_join_cg request). At state 2, if no net_commit_trans message comes before the timer T1 expires, the node will release all the resources reserved for the pending connection group, and go back to state 0. In the meanwhile, a net_abort_trans message is sent to the requester of net_join_cg to inform the failure of the transaction.

At state 1, if the connection group has not yet existed, the CM at the node calls the routing algorithm to find the next node towards the owner of the CG. The CM forwards the net_join_cg request to the next node. Then it goes to state 5 (waiting for a net_join_cg response back). If a negative net_join_cg response comes back or a time out occurs, the CM releases all the resources reserved and sends a negative net_join_cg response to the requester, and then it goes to state 0.

When a positive net_join_cg response is received at state 5, the CM goes to state 6 (waiting for commit request). If a time out occurs at state 6, the CM releases all the resources reserved and sends a net_abort_trans request to both the net_join_cg requester and the next_node, then it goes to state 0. If a commit_trans request is received, it forwards the commit request to the next_node, and then it goes to state 7 (waiting for commit_trans response).

When a positive commit_trans response is received at state 7, the CM commits all the reservations and sends a positive commit_trans response to the requester, then it goes to state 4. If a negative commit_trans response is received, the CM releases all the resources reserved and sends a negative commit_trans response to the requester, and it then goes to state 0. When a time out occurs at state 7, the CM releases all the resources reserved and sends a negative commit_trans response to the requestor and a net_abort_trans request to the next_node, and then it goes to state 0.

** Time out always makes confusion. No matter what actions taken when a timeout occurs, we may face some inconsistency in the network. The only way going around this is to make two assumptions: 1) the underlying transport protocol will correctly report the link status; 2) the timer is set long enough such that no timeout will occur for normal operations. If we make the second assumption, then we don't have to put the time issue in CMNP specification. Just let it be an implementation issue. We may mention this in the previous section. For the correctness proof, we can use the word "eventually". CMNP is not only a protocol, it also manages the resources. It has to interface with the lower level, say the transport layer. Do we have to consider the reactions when a link failure information is received as a part of CMNP specification?   dakang's comment **

6.3.3 NET_DROP_CG

Synopsis:

A NET_DROP_CG is initiated by an SM to drop an end-point from a CG. If the connection group indicated by the ***cg_id*** in the incoming NET_DROP_CG request does not exist at the current node, a negative response is sent back to the sender of the request with cg_cause set to CG_NOT_EXIST. If there is only one other edge evolved in the connection group at the current node, the resources for the connection group at the node will be marked as waiting-for-release, and a NET_DROP_CG request is forwarded along that edge. If the node has more than one other edges that are involved in the CG, an ACK is sent to the requester. When a COMMIT_TRANS is received, all the resources for that CG along the net_drop_cg path are released.

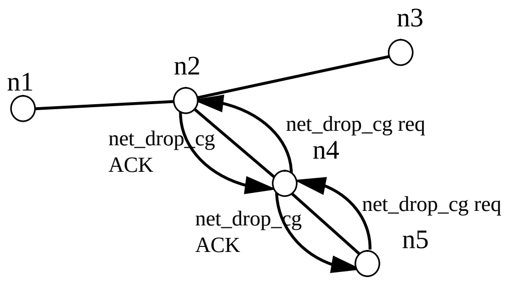


Figure 18. A NET_DROP_CG Example

Figure 18 shows an example of a successful drop_cg activity. Node n5 initiates the net_drop_cg request. Since there is only one other link in the CG at n4, n4 forwards the request to n2. At n2, there are more than one other links in the CG. So n2 stops the propagation of net_drop_cg request. It sends an ACK back. The commit requests and responses, which are not shown in the figure, will go the same way as the net_drop_cg requests and the responses. After the commitment, link (n2, n4) and (n4, n5) are removed from the CG.

Message Formats:

NET_DROP_CG request:

IE Type	Data Size (bytes)	Data	Mandatory or Optional	Default Value
OP_TYPE	1	NET_DROP_CG	M	
MULTIPHASE	1	REQUEST	M	
MSG_ID	2		M	
TRANS_ID	28		M	
SM_Requester	28		M	
CG_Owner	28		M	
CG_ID	28		M	
Send_REND	28		M	
Receive_REND	28		M	

NET_DROP_CG response:

IE Type	Data Size (bytes)	Data	Mandatory or Optional	Default Value
OP_TYPE	1	NET_DROP_CG	M	
MULTIPHASE	1	RESPONSE	M	
MSG_ID	2		M	
TRANS_ID	28		M	
SM_Requester	28		M	

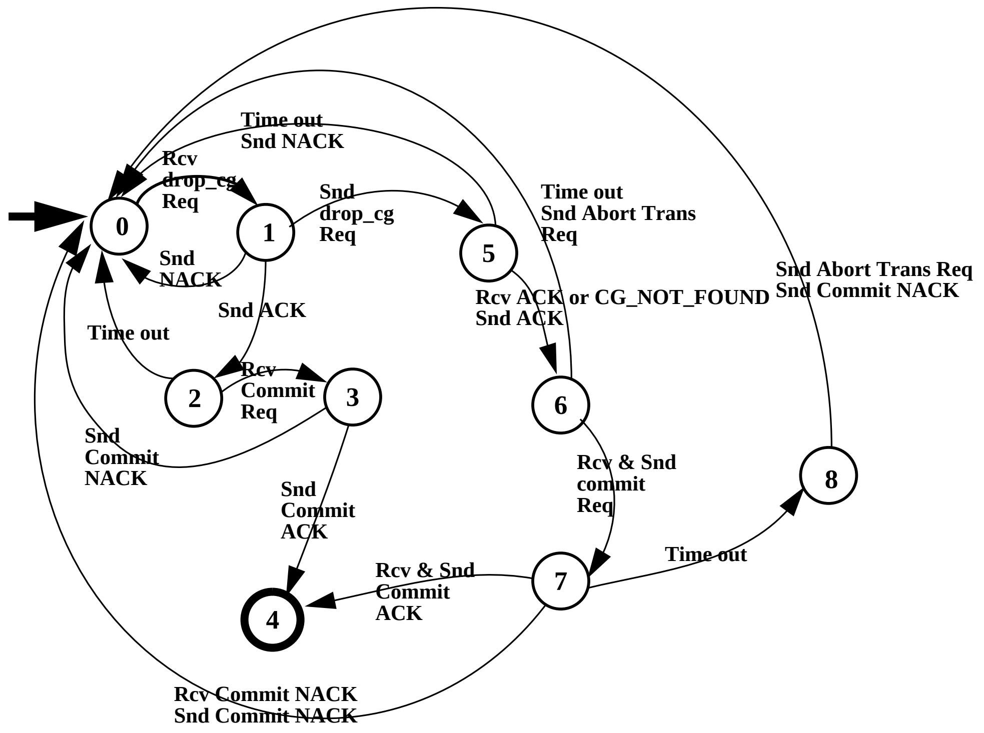
IE Type	Data Size (bytes)	Data	Mandatory or Optional	Default Value
CG_CAUSE	1		O	OK
CG_Owner	28		M	
CG_ID	28		M	
Send_REND	28		M	
Receive_REND	28		M	

Operation:

A node starts a net_drop_cg operation by receiving a net_drop_cg request. We refer the sender of the request as the requester.

Figure 19 is the state transition diagram of a CM at a node activated by receiving a NET_DROP_CG request.

Figure 19. net_drop_cg state transition



0 -- initial state
1 -- drop_cg request pending
2 -- waiting for commit request
3 -- commit request processing
4 -- successful drop_cg

5 -- waiting for drop_cg_response
6 -- waiting for commit request
7 -- waiting for commit response
8 -- commit response time out

State transitions start at state 0 (no net_drop_cg request has been received yet). When a net_drop_cg request is received, the node sets up a timer, then it goes to the state 1 (a net_drop_cg request pending). If there are more than two other edges are involved in the CG, an ACK is sent back, and then it goes to state 2, waiting for commit request. When the net_commit_trans request comes, the node release the resources for the edge and sends an ACK back, then it goes to state 4, the end of a successful net_drop_cg action. If the CG does not exist at the node, an NACK with CG_NOT_FOUND is sent back. If there is only one other edge in the CG, the CM will forward the net_drop_cg request along that edge, and it goes to state 5, waiting for the response.

6.3.4 NET_DESTROY_CG

Synopsis:

A NET_DESTROY_CG is initially sent by a CG's owner to request to release all the resources for a connection group. The NET_DESTROY_CG messages propagate along the connection tree to all the nodes involved in the CG. When committed, all the resources for that CG are released.

Message Formats:

NET_DESTROY_CG request:

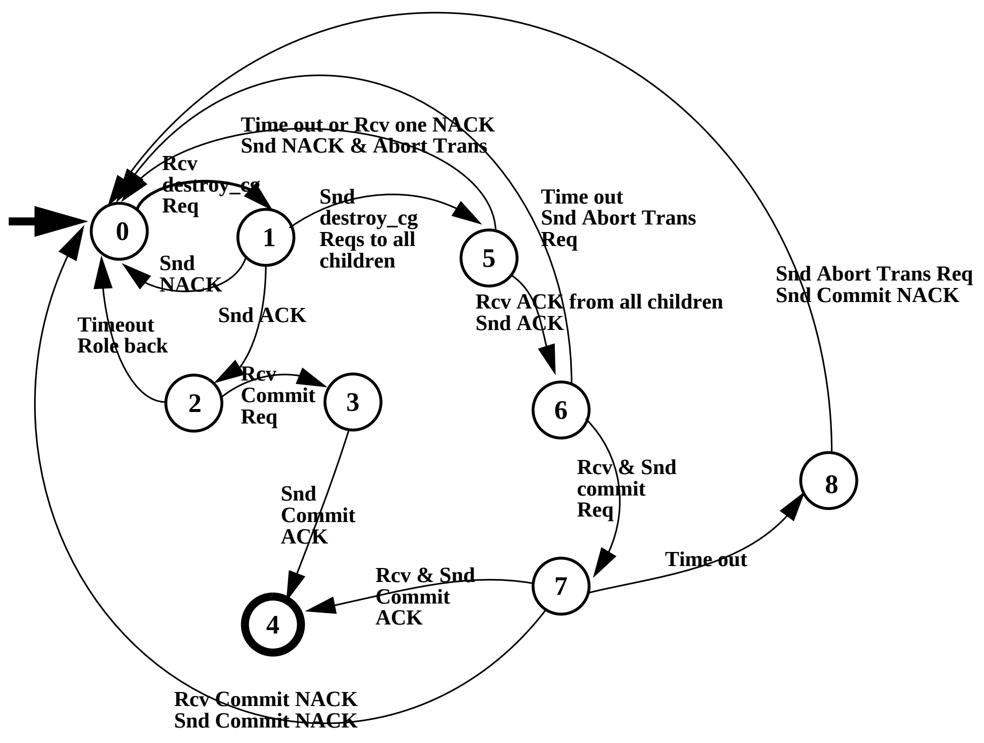
IE Type	Data Size (bytes)	Data	Mandatory or Optional	Default Value
OP_TYPE	1	NET_DESTROY_CG	M	
MULTIPHASE	1	REQUEST	M	
MSG_ID	2		M	
TRANS_ID	28		M	
SM_Requester	28		M	
CG_Owner	28		M	
CG_ID	28		M	
Send_REND	28		M	
Receive_REND	28		M	

NET_DESTROY_CG response:

IE Type	Data Size (bytes)	Data	Mandatory or Optional	Default Value
OP_TYPE	1	NET_DESTROY_CG	M	
MULTIPHASE	1	RESPONSE	M	
MSG_ID	2		M	
TRANS_ID	28		M	
SM_Requester	28		M	
CG_CAUSE	1		O	OK
CG_Owner	28		M	
CG_ID	28		M	
Send_REND	28		M	
Receive_REND	28		M	

Figure 20. destroy_cg transition diagram

Operation:



0 -- initial state
1 -- destroy_cg request pending
2 -- waiting for commit request
3 -- commit request processing
4 -- successful destroy_cg
5 -- waiting for destroy_cg_response
6 -- waiting for commit request
7 -- waiting for commit response
8 -- commit response time out

State transitions start at state 0, no net_destroy_cg request has been received yet. When a net_destroy_cg request is received, the node sets up a timer T1, then it goes to the state 1 (a net_destroy_cg_request pending). At state 1, the node checks the validity of the net_destroy_cg request. If the request is not a valid one, a negative net_destroy_cg response is sent to the requester with the CG_cause or con_cause fields setting to a particular reason of failure, and the node goes back to state 0.

If the request is valid and there are no edges in the connection group other than the edge where the request came from, the node will send a positive net_destroy_cg response to the requester, then it goes to state 2 (waiting for a net_commit_trans request). When the net_commit_trans request comes, it goes to state 3, where it releases all the resources for that CG and sends a positive response back, and then it goes to state 4 (the end state of a successful net_destroy_cg request). At state 2, if no net_commit_trans message comes before the timer T1 expires, the node will release all the resources for the connection group, and go back to state 0.

If there are edges other than the edge where the request came from in the connection group, the CM will send net_destroy_cg requests along all the edges, it then goes to state 5, waiting for all the acknowledgements back. When the CM receives acknowledgements from all the nodes to whom it has sent net_destroy_cg requests, it will send net_destroy_cg ack to the requester, then it goes to state 6, waiting for the commit_trans request. When the net_commit_trans request comes, it forwards the net_commit_trans request to all the nodes where it sent net_destroy_cg requests to. When it receives all the acknowledgements, it sends net_commit_trans ack to the requester, releases all the resources for the cg, and goes to state 4.

If a nack is received when the node is in state 5, the node will send nack to the requester and sends abort_trans to all the nodes to whom it has sent net_destroy_cg requests to abort the transaction.

6.3.5 NET_MOD_CG

Synopsis:

A NET_MOD_CG is initially sent by a CG's owner to request to modify CG attributes. The NET_MOD_CG messages propagate along the connection tree to all the nodes involved in the CG to modify the CG attributes.

Message Formats:

NET_MOD_CG request:

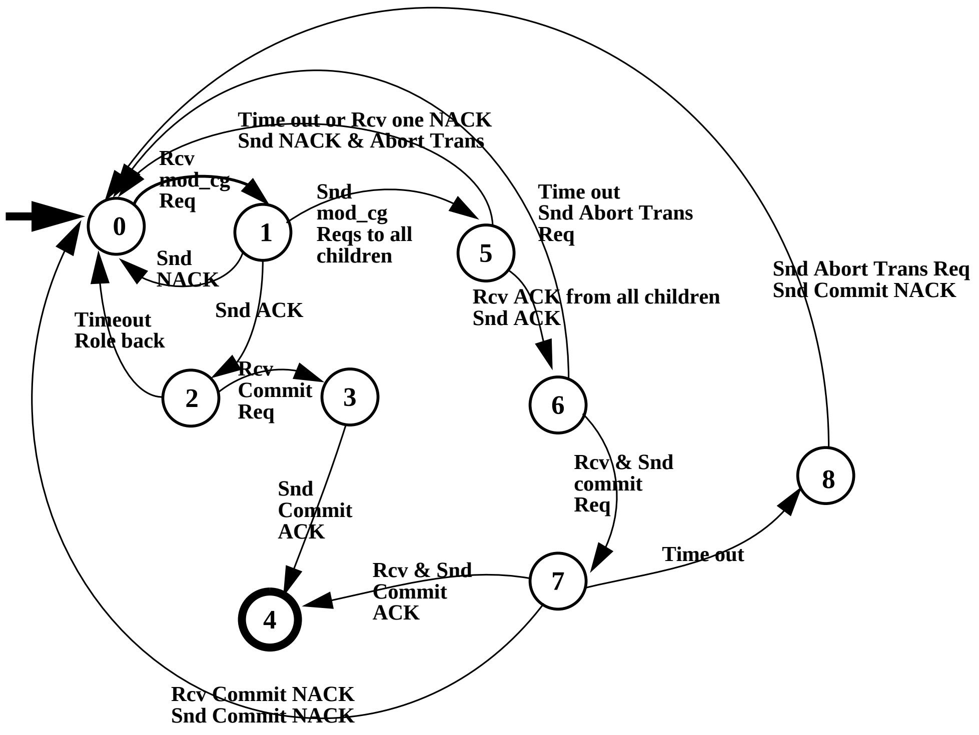
IE Type	Data Size (bytes)	Data	Mandatory or Optional	Default Value
OP_TYPE	1	NET_MOD_CG	M	
MULTIPHASE	1	REQUEST	M	
MSG_ID	2		M	
TRANS_ID	28		M	
SM_Requester	28		M	
CG_Owner	28		M	
CG_ID	28		M	
Send_REND	28		M	
Receive_REND	28		M	
CG_ACK_FLAG	1		O	OFF
CORRELATION	1	STRONG	O	STRONG
TRUNKING	1	YES	O	YES

NET_MOD_CG response:

IE Type	Data Size (bytes)	Data	Mandatory or Optional	Default Value
OP_TYPE	1	NET_MOD_CG	M	
MULTIPHASE	1	RESPONSE	M	
MSG_ID	2		M	
TRANS_ID	28		M	
SM_Requester	28		M	
CG_CAUSE	1		O	OK
CG_Owner	28		M	
CG_ID	28		M	
Send_REND	28		M	
Receive_REND	28		M	

Operation:

Figure 21. net_mod_cg transition diagram


0
1
2
3
4
5
6
7
8
Rcv mod_cg Req
Snd NACK
Snd mod_cg Reqs to all children
Time out or Rcv one NACK Snd NACK & Abort Trans
Time out Snd Abort Trans Req
Snd Abort Trans Req Snd Commit NACK
Rcv ACK from all children Snd ACK
Snd ACK
Timeout Role back
Rcv Commit Req
Snd Commit ACK
Rcv & Snd commit Req
Time out
Rcv & Snd Commit ACK
Rcv Commit NACK Snd Commit NACK

0 -- initial state	**5 -- waiting for mod_cg_response**
1 -- mod_cg request pending	**6 -- waiting for commit request**
2 -- waiting for commit request	**7 -- waiting for commit response**
3 -- commit request processing	**8 -- commit response time out**
4 -- successful mod_cg	

State transitions start at state 0, no net_mod_cg request has been received yet. When a net_mod_cg request is received, the node sets up a timer T1, then it goes to the state 1 (a net_mod_cg_request pending). At state 1, the node checks the validity of the net_mod_cg request. If the request is not a valid one, a negative net_mod_cg response is sent to the requester with the cg_cause or con_cause fields setting to a particular reason of failure, and the node goes back to state 0.

If the request is valid and there are no edges in the connection group other than the edge where the request came from, the node will send a positive net_mod_cg response to the requester, then it goes to state 2 (waiting for a net_commit_trans request). When the net_commit_trans request comes, it goes to state 3, where it make all the modifications permenant for that CG and sends a positive response back, and then it goes to state 4 (the end state of a successful net_mod_cg request). At state 2, if no net_commit_trans message comes before the timer T1 expires, the node will abort all the modifications, and go back to state 0.

If there are edges other than the edge where the request came from in the connection group, the CM will send net_mod_cg requests along all the edges, it then goes to state 5, waiting for all the acknowledgements. When the CM receives acknowledgements from all the nodes to whom it has sent net_mod_cg requests, it will send net_mod_cg ack to the requester, then it goes to state 6, waiting for the commit_trans request. When the net_commit_trans request comes, it forwards the net_commit_trans request to all the nodes where it sent net_mod_cg requests to. When it receives all the acknowledgements, it sends net_commit_trans ack to the requester, makes all the modifications permenent for the cg, and goes to state 4.

If a nack is received when the node is in state 5, the node will send nack to the requester and sends abort_trans to all the nodes to whom it has sent net_mod_cg requests to abort the transaction.

6.3.6 NET_QUERY_CG_OWNER

Synopsis:

When a NET_JOIN_CG, or NET_DROP_CG request is received and the current node has the CG_Ack_Flag set, the node has to query the CG owner to get the permission to add or drop a CG at the node. A NET_QUERY_CG_OWNER message serves this purpose. When a NET_QUERY_CG_OWNER acknowledgement is received, the previous operation can continue. Otherwise, an NACK will be sent with CG_OWNER_DISAPPROVE in CG_cause. NET_QUERY_CG_OWNER does not relate to any resource allocation or deallocation, so that it will not be included in a transaction.

Message Formats:

NET_QUERY_CG_OWNER request:

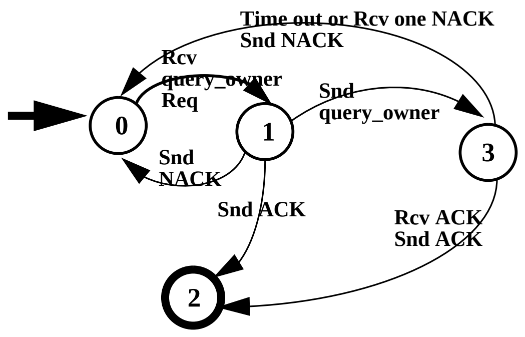
IE Type	Data Size (bytes)	Data	Mandatory or Optional	Default Value
OP_TYPE	1	NET_QUERY_CG_OWNER	M	
MULTIPHASE	1	REQUEST	M	
MSG_ID	2		M	
TRANS_ID	28		M	
SM_Requester	28		M	
CG_Owner	28		M	
CG_ID	28		M	
Send_REND	28		M	
Receive_REND	28		M	
CG_ACK_FLAG	1		O	OFF
CORRELATION	1	STRONG	O	STRONG
TRUNKING	1	YES	O	YES

NET_QUERY_CG_OWNER response:

IE Type	Data Size (bytes)	Data	Mandatory or Optional	Default Value
OP_TYPE	1	NET_QUERY_CG_OWNER	M	
MULTIPHASE	1	RESPONSE	M	
MSG_ID	2		M	
TRANS_ID	28		M	
SM_Requester	28		M	
CG_CAUSE	1		O	OK
CG_Owner	28		M	
CG_ID	28		M	
Send_REND	28		M	
Receive_REND	28		M	

Operation:

Figure 22. net_query_cg_owner transition diagram



0 -- initial state
1 -- query_cg_owner request pending
2 -- successful query_cg_owner
3 -- waiting for response

State transitions start at state 0, no net_query_cg_owner request has been received yet. When a net_query_cg_owner request is received, the node sets up a timer T1, then it goes to the state 1 (a net_query_cg_owner_request pending). If the current node is the owner of the cg, it sends ACK or Nack based on its decision. If it is not the owner, the will forward the request towards the owner. Then it will pass the response to the requester. It a timeout occurs when it’s waiting for the response, it will send a negative response to the requester.

6.3.7 NET_ADD_CON

Synopsis:

A NET_ADD_CON request is a message initiated by a SM to request to add one or more new connections into an existing CG. When a NET_ADD_CON request is received, the CM checks the resources. If the resources required are available, the CM will reserve the resources for the new connections, and send NET_ADD_CON to all other CMs who are in the CG and have an edge to the current node. When the CM receives all ACKs from the CMs whom it has sent requests to, it sends an ACK to the sender of the NET_ADD_CON. When a NET_COMMIT_TRANS is received, all the reserved resources for the new connections become permanent.

Message Formats:

NET_ADD_CON request:

IE Type	Data Size (bytes)	Data	Mandatory or Optional	Default Value
OP_TYPE	1	NET_ADD_CON	M	
MULTIPHASE	1	REQUEST	M	
MSG_ID	2		M	
TRANS_ID	28		M	
SM_Requester	28		M	
CG_Owner	28		M	
CG_ID	28		M	
Send_REND	28		M	
Receive_REND	28		M	
CON_ID	1		M	
CON_OWNER	28		M	
TxBW	12		M	
RxBW	12		M	
ACK_FLAG	1		O	OFF
CON_TYPE	1		O	VC
QOS	1		O	HIGH
MAPPING	1			
NOTIFY	28			
TxVPI	1			
TxVCI	2			
RxVPI	1			
RxVCI	2			
CON_ID	1			
		...		

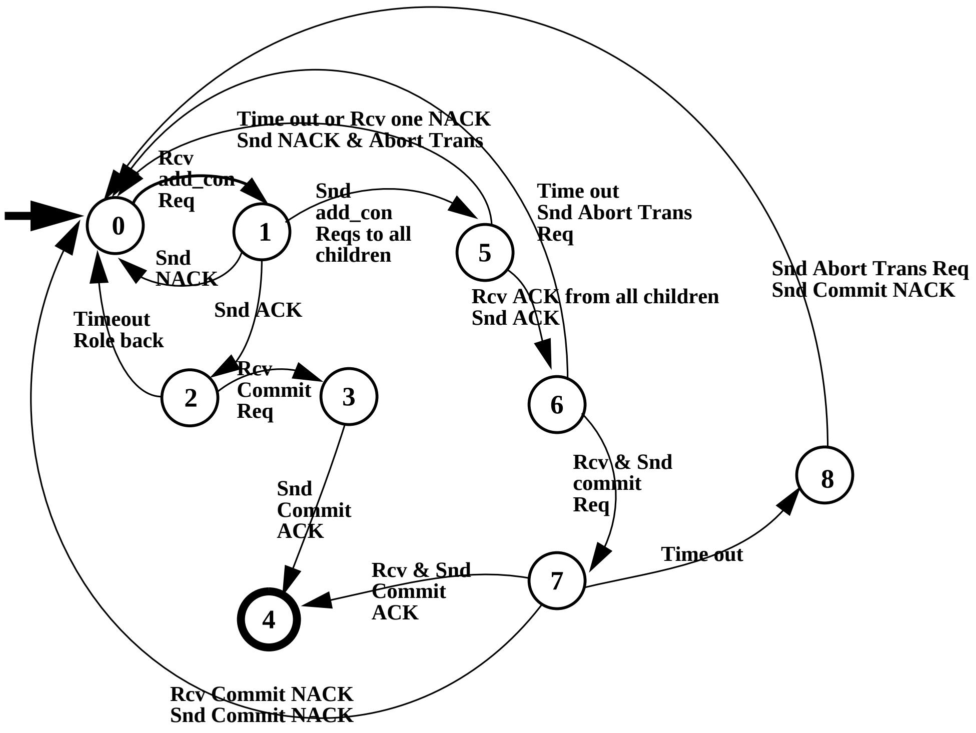
NET_ADD_CON response:

IE Type	Data Size (bytes)	Data	Mandatory or Optional	Default Value
OP_TYPE	1	NET_ADD_CON	M	
MULTIPHASE	1	RESPONSE	M	
MSG_ID	2		M	
TRANS_ID	28		M	
SM_Requester	28		M	
CG_CAUSE	1		O	OK
CG_Owner	28		M	
CG_ID	28		M	
Send_REND	28		M	
Receive_REND	28		M	
CON_CONTEXT	1		M	
CON_CAUSE	1		O	OK
TxVPI	1			
TxVCI	2			
RxVPI	1			

IE Type	Data Size (bytes)	Data	Mandatory or Optional	Default Value
RxVCI	2			

Operation:

Figure 23. net_add_con transition diagram


0
1
2
3
4
5
6
7
8
Rcv add_con Req
Snd NACK
Snd add_con Reqs to all children
Time out or Rcv one NACK Snd NACK & Abort Trans
Time out Snd Abort Trans Req
Snd Abort Trans Req Snd Commit NACK
Rcv ACK from all children Snd ACK
Snd ACK
Timeout Role back
Rcv Commit Req
Snd Commit ACK
Rcv & Snd commit Req
Time out
Rcv & Snd Commit ACK
Rcv Commit NACK Snd Commit NACK

0 -- initial state	**5 -- waiting for add_con_response**
1 -- add_con request pending	**6 -- waiting for commit request**
2 -- waiting for commit request	**7 -- waiting for commit response**
3 -- commit request processing	**8 -- commit response time out**
4 -- successful add_con	

State transitions start at state 0, no net_add_con request has been received yet. When a net_add_con request is received, the node sets up a timer T1, then it goes to the state 1 (a net_add_con_request pending). At state 1, the node checks its resources to see if the request can be satisfied. If the request is no enogh resources to support the connection, a negative net_add_conadd_con response is sent to the requester with the con_cause fields setting to the reason of failure, and the node goes back to state 0.

If there is enough resources, it reserves the resources. If there are no edges in the connection group other than the edge where the request came from, the node will send a positive net_add_con response to the requester, then it goes to state 2 (waiting for a net_commit_trans request). When the net_commit_trans request comes, it goes to state 3, where it make all the reservation permenant for that connection and sends a positive response back, and then it goes to state 4 (the end state of a successful net_add_con request). At state 2, if no net_commit_trans message comes before the timer T1 expires, the node will abort all the transaction, and go back to state 0.

If there are edges other than the edge where the request came from in the connection group, the CM will send net_add_con requests along all the edges, it then goes to state 5, waiting for acknowledgements. When the CM receives acknowledgements from all the nodes to whom it has sent net_add_con requests, it will send net_add_con ack to the requester, then it goes to state 6, waiting for the commit_trans request. When the net_commit_trans request comes, it forwards the net_commit_trans request to all the nodes where it sent net_add_con requests to. When it receives all the acknowledgements, it sends net_commit_trans ack to the requester, makes the reservation permenent for the connection, and goes to state 4.

If a nack is received when the node is in state 5, the node will send nack to the requester and sends abort_trans to all the nodes to whom it has sent net_add_con requests to abort the transaction.

6.3.8 NET_JOIN_CON

Synopsis:

A NET_JOIN_CON request is a message initiated by a SM to request to join an existing connection. When a NET_JOIN_CON request is received, the CM checks if the connection exists at the current node. If the connection does exist, it will send an ACK to the sender. If the connection does not exist, the CM will forward the message towards the owner of the connection and make the resource reservation temporarily. When a NET_COMMIT_TRANS is received, all the reservations become permanent.

Message Formats:

NET_JOIN_CON request:

IE Type	Data Size (bytes)	Data	Mandatory or Optional	Default Value
OP_TYPE	1	NET_JOIN_CON	M	
MULTIPHASE	1	REQUEST	M	
MSG_ID	2		M	
TRANS_ID	28		M	
SM_Requester	28		M	
CG_Owner	28		M	
CG_ID	28		M	
Send_REND	28		M	
Receive_REND	28		M	
CON_ID	1		M	
CON_OWNER	28		M	
TxBW	12		M	
RxBW	12		M	
ACK_FLAG	1		O	OFF
CON_TYPE	1		O	VC
QOS	1		O	HIGH
MAPPING	1			
NOTIFY	28			
TxVPI	1			
TxVCI	2			
RxVPI	1			
RxVCI	2			
CON_ID	1			
		...		

NET_JOIN_CON response:

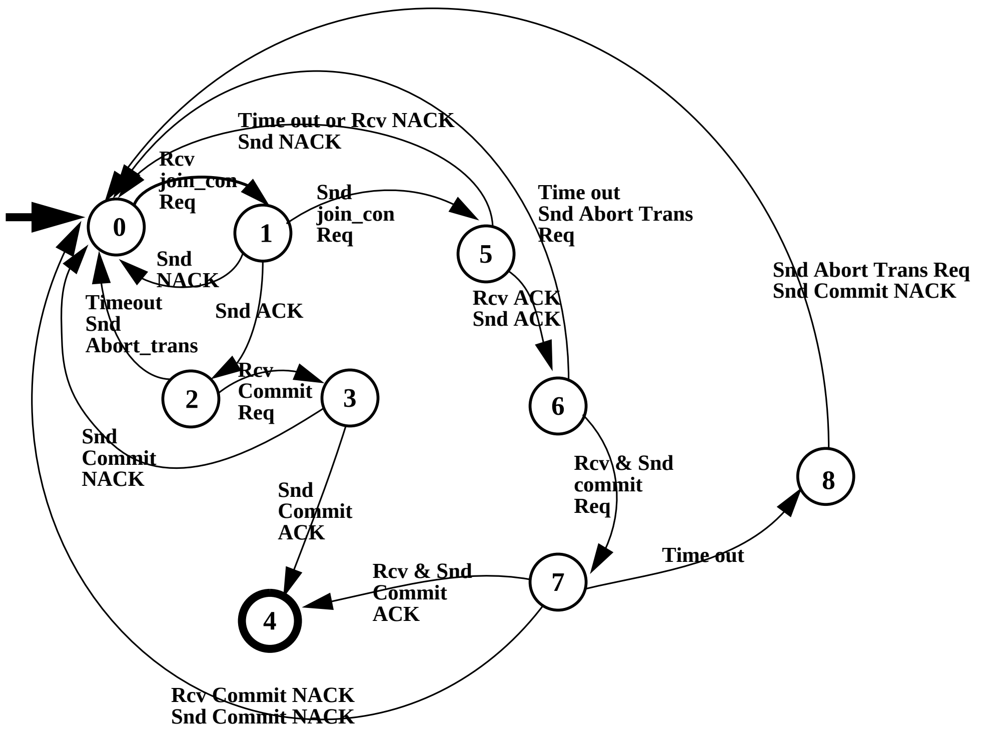
IE Type	Data Size (bytes)	Data	Mandatory or Optional	Default Value
OP_TYPE	1	NET_JOIN_CON	M	
MULTIPHASE	1	RESPONSE	M	
MSG_ID	2		M	
TRANS_ID	28		M	
SM_Requester	28		M	
CG_CAUSE	1		O	OK
CG_Owner	28		M	
CG_ID	28		M	
Send_REND	28		M	
Receive_REND	28		M	
CON_CONTEXT	1		M	
CON_CAUSE	1		O	OK
TxVPI	1			
TxVCI	2			
RxVPI	1			
RxVCI	2			

Operation:

A node starts a net_join_con operation by receiving a net_join_con request. We refer the sender of the request as requester. Included in the request are a cg_id, a set of connection group parameters, and a set of connection parameters. Based on the cg_ack_flag and ack_flags in connection objects, the state transition diagram can be different.

Figure 16. is the state transition diagram of a CM at a node activated by receiving a NET_join_con request where all the cg_ack_flag and ack_flags are off.

Figure 24. net_join_con state transition diagram



0 -- initial state
1 -- join_con request pending
2 -- waiting for commit request
3 -- commit request processing
4 -- successful join_con

5 -- waiting for join_con_response
6 -- waiting for commit request
7 -- waiting for commit response
8 -- commit response time out

State transitions start at state 0, no net_join_con request has been received yet. When a net_join_con request is received, the node sets up a timer T1, then it goes to the state 1 (a net_join_con_request pending). At state 1, the node checks its resources to see if the request can be satisfied at the current node. If there is enough spare bandwidth on the link where the request came from, it reserves the bandwidth on that link. Otherwise, a negative net_join_con response is sent to the requester, and the node goes back to state 0.

If the connections indicated by the con_ids in the net_join_con message have already existed at the node and there are enough resources available at the current node to satisfy the request, the node will send a positive net_join_con response to the requester, then it goes to state 2 (waiting for a net_commit_trans request). When the net_commit_trans request comes, it goes to state 3, where it sends a positive response back, and then it goes to state 4 (the end state of a successful net_join_con request). At state 2, if no net_commit_trans message comes before the timer T1 expires, the node will release all the resources reserved for the pending connections, and go back to state 0. In the meanwhile, a net_abort_trans message is sent to the requester of net_join_con to inform the failure of the transaction.

At state 1, if the connection group has not yet existed, the CM finds the next node towards the owner of the CG. The CM forwards the net_join_con request to the next node. Then it goes to state 5 (waiting for a net_join_con response back). If a negative net_join_con response comes back or a time out occurs, the CM releases all the resources reserved and sends a negative net_join_con response to the requester, and then it goes to state 0.

When a positive net_join_con response is received at state 5, the CM goes to state 6 (waiting for commit request). If a time out occurs at state 6, the CM releases all the resources reserved and sends a net_abort_trans request to both the net_join_con requester and the next_node, then it goes to state 0. If a commit_trans request is received, it forwards the commit request to the next_node, and then it goes to state 7 (waiting for commit_trans response).

When a positive commit_trans response is received at state 7, the CM commits all the reservations made and sends a positive commit_trans response to the requester, then it goes to state 4. If a negative commit_trans response is received, the CM releases all the resources reserved and sends a negative commit_trans response to the requester, and it then goes to state 0. When a time out occurs at state 7, the CM releases all the resources reserved and sends a negative commit_trans response to the requester and a net_abort_trans request to the next_node, and then it goes to state 0.

6.3.9 NET_DROP_CON

Synopsis:

A NET_DROP_CON request is initiated by an SM to drop an end-point from a connection. If the connection does not exist at the current node, a negative response is sent back to the sender of the requester with con_cause set to CON_NOT_EXIST. If there is only one other edge evolved in the connection group, the resources for the connection group at the node will be marked as waiting-for-release, and a NET_DROP_CON request is forwarded along that edge. When a NET_COMMIT_TRANS is received, all the resources for the connection is released.

Message Formats:

NET_DROP_CON request:

IE Type	Data Size (bytes)	Data	Mandatory or Optional	Default Value
OP_TYPE	1	NET_DROP_CON	M	
MULTIPHASE	1	REQUEST	M	
MSG_ID	2		M	
TRANS_ID	28		M	
SM_Requester	28		M	
CG_Owner	28		M	
CG_ID	28		M	
Send_REND	28		M	
Receive_REND	28		M	
CON_ID	1		M	
CON_ID	1			
		...		

NET_DROP_CON response:

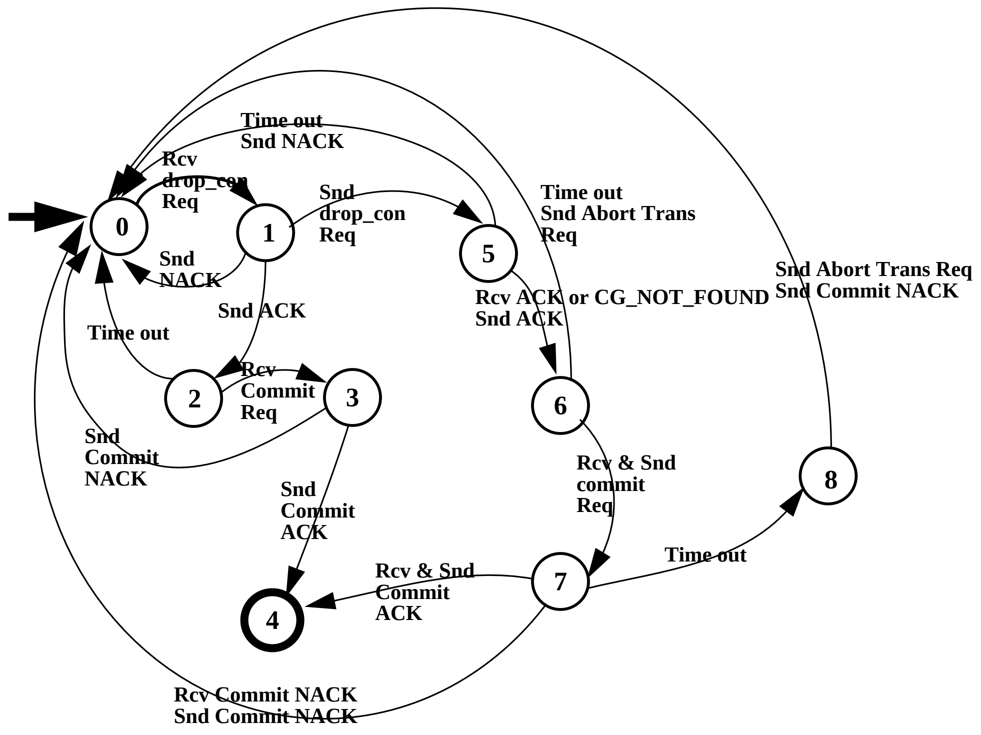
IE Type	Data Size (bytes)	Data	Mandatory or Optional	Default Value
OP_TYPE	1	NET_DROP_CON	M	
MULTIPHASE	1	RESPONSE	M	
MSG_ID	2		M	
TRANS_ID	28		M	
SM_Requester	28		M	
CG_CAUSE	1		O	OK
CG_Owner	28		M	
CG_ID	28		M	
Send_REND	28		M	
Receive_REND	28		M	
CON_ID	1		M	
CON_CAUSE	1		O	OK

Operation:

A node starts a net_drop_con operation by receiving a net_drop_con request. We refer the sender of the request as the requester.

Figure 25 is the state transition diagram of a CM at a node activated by receiving a net_drop_con request.

Figure 25. net_drop_con state transition



0 -- initial state	5 -- waiting for drop_con_response
1 -- drop_con request pending	6 -- waiting for commit request
2 -- waiting for commit request	7 -- waiting for commit response
3 -- commit request processing	8 -- commit response time out
4 -- successful drop_con	

State transitions start at state 0 (no net_drop_con request has been received yet). When a net_drop_con request is received, the node sets up a timer, then it goes to the state 1 (a net_drop_con request pending). If there are more than two other edges are involved in the CG, an ACK is sent back, and then it goes to state 2, waiting for commit request. When the net_commit_trans request comes, the node release the resources for the edge and sends an ACK back, then it goes to state 4, the end of a successful net_drop_con action. If the CG does not exist at the node, an NACK with CG_NOT_FOUND is sent back. If there is only one other edge in the CG, the CM will forward the net_drop_con request along that edge, and it goes to state 5, waiting for the response.

6.3.10 NET_DESTROY_CON

Synopsis:

A NET_DESTROY_CON is initiated by the connection owner SM to request to release all the resources for a connection. The NET_DESTROY_CON messages propagate along the connection tree to all the nodes involved in the CG. When committed, all the resources for the connection are released.

Message Formats:

NET_DESTROY_CON request:

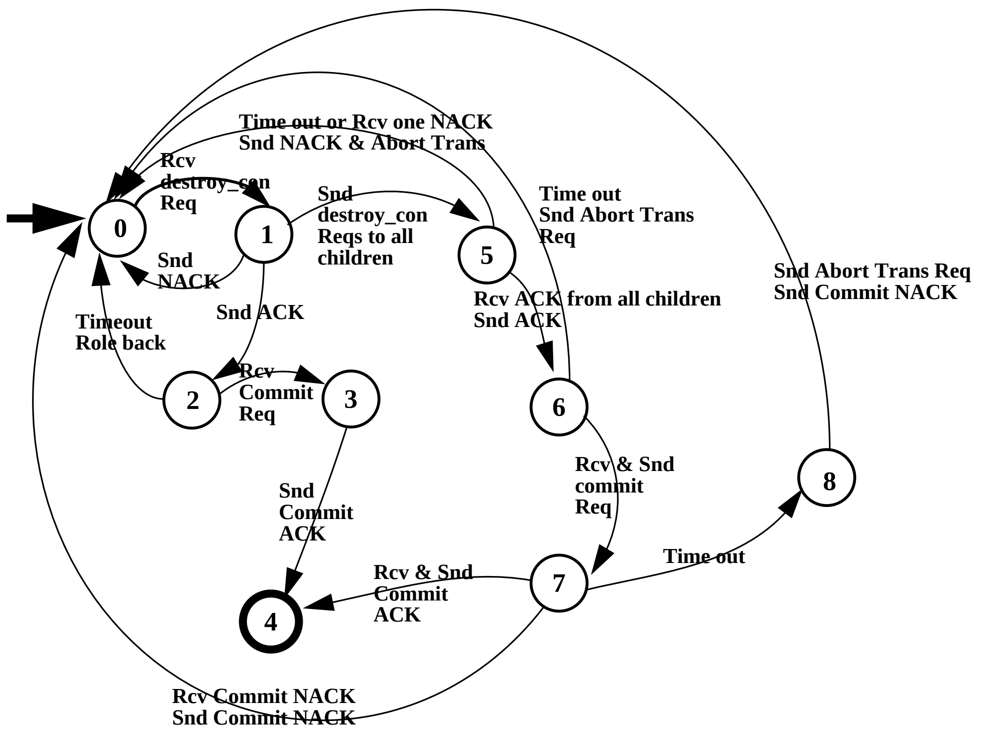
IE Type	Data Size (bytes)	Data	Mandatory or Optional	Default Value
OP_TYPE	1	NET_DESTROY_CON	M	
MULTIPHASE	1	REQUEST	M	
MSG_ID	2		M	
TRANS_ID	28		M	
SM_Requester	28		M	
CG_CAUSE	1		O	OK
CG_Owner	28		M	
CG_ID	28		M	
Send_REND	28		M	
Receive_REND	28		M	
CON_ID	1		M	
CON_CAUSE	1		O	OK

NET_DESTROY_CON response:

IE Type	Data Size (bytes)	Data	Mandatory or Optional	Default Value
OP_TYPE	1	NET_DESTROY_CON	M	
MULTIPHASE	1	RESPONSE	M	
MSG_ID	2		M	
TRANS_ID	28		M	
SM_Requester	28		M	
CG_CAUSE	1		O	OK
CG_Owner	28		M	
CG_ID	28		M	
Send_REND	28		M	
Receive_REND	28		M	
CON_ID	1		M	
CON_CAUSE	1		O	OK

Operation:

Figure 26. destroy_con transition diagram



0 -- initial state
1 -- destroy_con request pending
2 -- waiting for commit request
3 -- commit request processing
4 -- successful destroy_con

5 -- waiting for destroy_con_response
6 -- waiting for commit request
7 -- waiting for commit response
8 -- commit response time out

State transitions start at state 0, no net_destroy_con request has been received yet. When a net_destroy_con request is received, the node sets up a timer T1, then it goes to the state 1 (a net_destroy_con_request pending). At state 1, the node checks the validity of the net_destroy_con request. If the request is not a valid one, a negative net_destroy_con response is sent to the requester with the CG_cause or con_cause fields setting to a particular reason of failure, and the node goes back to state 0.

If the request is valid and there are no edges in the connection group other than the edge where the request came from, the node will send a positive net_destroy_con response to the requester, then it goes to state 2 (waiting for a net_commit_trans request). When the net_commit_trans request comes, it goes to state 3, where it releases all the resources for that connection and sends a positive response back, and then it goes to state 4 (the end state of a successful net_destroy_con request). At state 2, if no net_commit_trans message comes before the timer T1 expires, the node will release all the resources for the connection, and go back to state 0.

If there are edges other than the edge where the request came from in the connection group, the CM will send net_destroy_con requests along all the edges, it then goes to state 5, waiting for all the acknowledgements back. When the CM receives acknowledgements from all the nodes to whom it has sent net_destroy_con requests, it will send a net_destroy_con ack to the requester, then it goes to state 6, waiting for the commit_trans request. When the net_commit_trans request comes, it forwards the net_commit_trans request to all the nodes where it sent net_destroy_con requests to. When it receives all the acknowledgements, it sends net_commit_trans ack to the requester, releases all the resources for the connection, and goes to state 4.

If a nack is received when the node is in state 5, the node will send nack to the requester and sends abort_trans to all the nodes to whom it has sent net_destroy_con requests to abort the transaction.

6.3.11 NET_MOD_CON

Synopsis:

A NET_MOD_CON request is initiated by a connection owner SM to request to modify the attributes for the connections. The NET_MOD_CON messages propagate along the connection tree to all the nodes involved in the connection. When committed, all the modifications become permanent.

Message Formats:

NET_MOD_CON request:

IE Type	Data Size (bytes)	Data	Mandatory or Optional	Default Value
OP_TYPE	1	NET_MOD_CON	M	
MULTIPHASE	1	REQUEST	M	
MSG_ID	2		M	
TRANS_ID	28		M	
SM_Requester	28		M	
CG_Owner	28		M	
CG_ID	28		M	
Send_REND	28		M	
Receive_REND	28		M	
CON_ID	1		M	
CON_OWNER	28		M	
TxBW	12		M	
RxBW	12		M	
ACK_FLAG	1		O	OFF
CON_TYPE	1		O	VC
QOS	1		O	HIGH
MAPPING	1			
NOTIFY	28			
TxVPI	1			
TxVCI	2			
RxVPI	1			
RxVCI	2			
CON_ID	1			
		...		

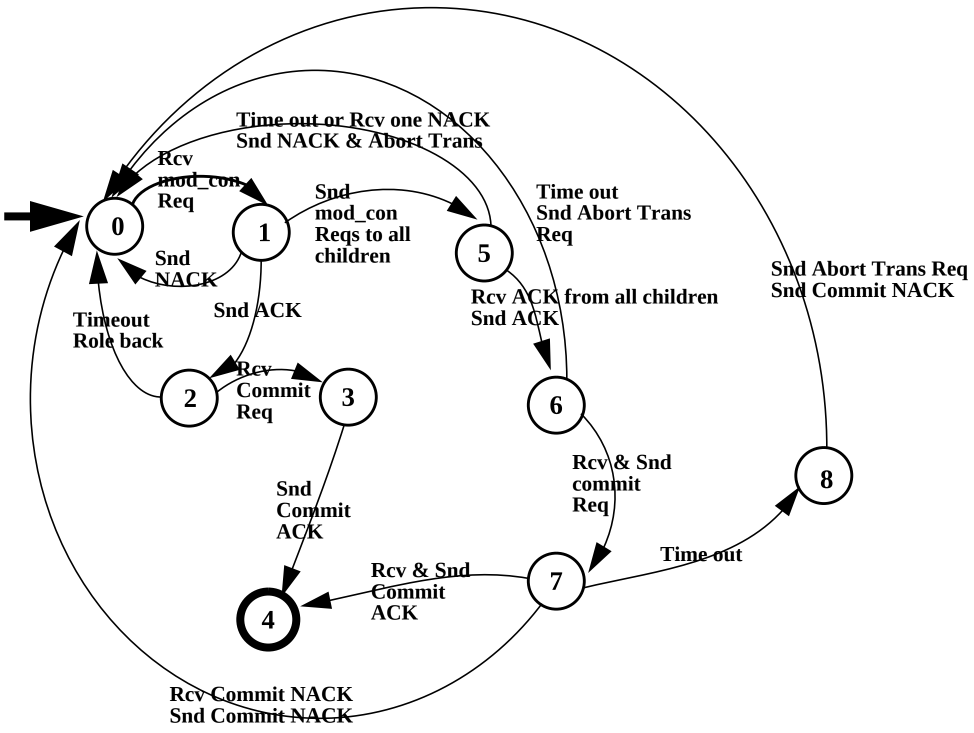
NET_MOD_CON response:

IE Type	Data Size (bytes)	Data	Mandatory or Optional	Default Value
OP_TYPE	1	NET_MOD_CON	M	
MULTIPHASE	1	RESPONSE	M	
MSG_ID	2		M	
TRANS_ID	28		M	
SM_Requester	28		M	
CG_CAUSE	1		O	OK
CG_Owner	28		M	
CG_ID	28		M	
Send_REND	28		M	
Receive_REND	28		M	
CON_CONTEXT	1		M	
CON_CAUSE	1		O	OK
TxVPI	1			
TxVCI	2			

IE Type	Data Size (bytes)	Data	Mandatory or Optional	Default Value
RxVPI	1			
RxVCI	2			

Operation:

Figure 27. net_mod_con transition diagram



0 -- initial state
1 -- mod_con request pending
2 -- waiting for commit request
3 -- commit request processing
4 -- successful mod_con
5 -- waiting for mod_cg_response
6 -- waiting for commit request
7 -- waiting for commit response
8 -- commit response time out

State transitions start at state 0, no net_mod_con request has been received yet. When a net_mod_con request is received, the node sets up a timer T1, then it goes to the state 1 (a net_mod_con_request pending). At state 1, the node checks the validity of the net_mod_con request and its resources. If the request is not a valid one or the resources at the current node does not allow the modifications, a negative net_mod_con response is sent to the requester with the cg_cause or con_cause fields setting to a particular reason of failure, and the node goes back to state 0.

When the request is valid and there exists enough resources to support the modifications, if there are no edges in the connection group other than the edge where the request came from, the node will send a positive net_mod_con response to the requester and reserve the resources required, then it goes to state 2 (waiting for a net_commit_trans request). When the net_commit_trans request comes, it goes to state 3, where it make all the modifications permenant for the connections and sends a positive response back, and then it goes to state 4 (the end state of a successful net_mod_con request). At state 2, if no net_commit_trans message comes before the timer T1 expires, the node will abort all the modifications, and go back to state 0.

If there are edges other than the edge where the request came from in the connection group, the CM will send net_mod_con requests along all the edges, it then goes to state 5, waiting for all the acknowledgements. When the CM receives acknowledgements from all the nodes to whom it has sent net_mod_con requests, it will send net_mod_con ack to the requester, then it goes to state 6, waiting for the commit_trans request. When the net_commit_trans request comes, it forwards the net_commit_trans request to all the nodes where it has sent net_mod_con requests to. When it receives all the acknowledgements, it sends net_commit_trans ack to the requester, makes all the modifications permenent for the connections, and goes to state 4.

If a nack is received when the node is in state 5, the node will send nack to the requester and sends abort_trans to all the nodes to whom it has sent net_mod_con requests to abort the transaction.

6.3.12 NET_QUERY_CON_OWNER

Synopsis:

When a NET_JOIN_CON, or NET_DROP_CON request is received and the current node has the CON_Ack_-Flag set, the node has to query the connection owner to get the permission to add or drop a connection at the node. A NET_QUERY_CON_OWNER message serves this purpose. When a NET_QUERY_CON_OWNER acknowledgement is received, the previous operation can continue. Otherwise, an NACK will be sent with CG_OWNER_DISAPPROVE as the failure cause. NET_QUERY_CON_OWNER does not initiated by an SM, so that it will not be included in a transaction.

Message Formats:

NET_QUERY_CON_OWNER request:

IE Type	Data Size (bytes)	Data	Mandatory or Optional	Default Value
OP_TYPE	1	NET_QUERY_CON_OWNER	M	
MULTIPHASE	1	REQUEST	M	
MSG_ID	2		M	
TRANS_ID	28		M	
SM_Requester	28		M	
CG_Owner	28		M	
CG_ID	28		M	
Send_REND	28		M	
Receive_REND	28		M	
CON_ID	1		M	
CON_OWNER	28		M	
TxBW	12		M	
RxBW	12		M	
ACK_FLAG	1		O	OFF
CON_TYPE	1		O	VC
QOS	1		O	HIGH
MAPPING	1			
NOTIFY	28			
TxVPI	1			
TxVCI	2			
RxVPI	1			
RxVCI	2			
CON_ID	1			
		...		

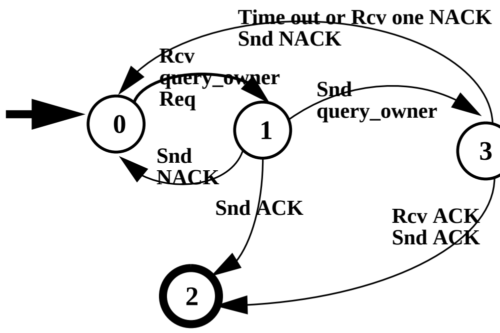
NET_QUERY_CON_OWNER response:

IE Type	Data Size (bytes)	Data	Mandatory or Optional	Default Value
OP_TYPE	1	NET_QUERY_CON_OWNER	M	
MULTIPHASE	1	RESPONSE	M	
MSG_ID	2		M	
TRANS_ID	28		M	
SM_Requester	28		M	
CG_CAUSE	1		O	OK
CG_Owner	28		M	
CG_ID	28		M	
Send_REND	28		M	
Receive_REND	28		M	

IE Type	Data Size (bytes)	Data	Mandatory or Optional	Default Value
CON_CONTEXT	1		M	
CON_CAUSE	1		O	OK
TxVPI	1			
TxVCI	2			
RxVPI	1			
RxVCI	2			

Operation:

Figure 28. net_query_con_owner transition diagram



0 -- initial state
1 -- query_con_owner request pending
2 -- successful query_cg_owner
3 -- waiting for response

State transitions start at state 0, no net_query_con_owner request has been received yet. When a net_query_con_owner request is received, the node sets up a timer T1, then it goes to the state 1 (a net_query_con_owner_request pending). If the current node is the owner of the cg, it sends ACK or Nack based on its decision. If it is not the owner, the will forward the request towards the connection owner. Then it will pass the response to the requester. It a timeout occurs when it’s waiting for the response, it will send a negative response to the requester.

NET_COMMIT_TRANS

Synopsis:

When a NET_COMMIT_TRANS request is received and the transaction exists at the current CM, the CM passes the NET_COMMIT_TRANS to all the neighbor CMs that are participated in the transaction. When ACKs are received from all of them, the CM commits the transaction and sends the ACK back to the sender of the NET_COMMIT_-TRANS requester.

** Since a transaction can contain multiple operation requests, it is possible a transaction commit can go into a cycle. To avoid infinite loop, commit_trans only take effect when the first time it is received **

Message Formats:

NET_COMMIT_TRANS request:

IE Type	Data Size (bytes)	Data	Mandatory or Optional	Default Value
OP_TYPE	1	NET_COMMIT_TRANS	M	
MULTIPHASE	1	REQUEST	M	
MSG_ID	2		M	
TRANS_ID	28		M	
SM_Requester	28		M	
Send_REND	28		M	
Receive_REND	28		M	

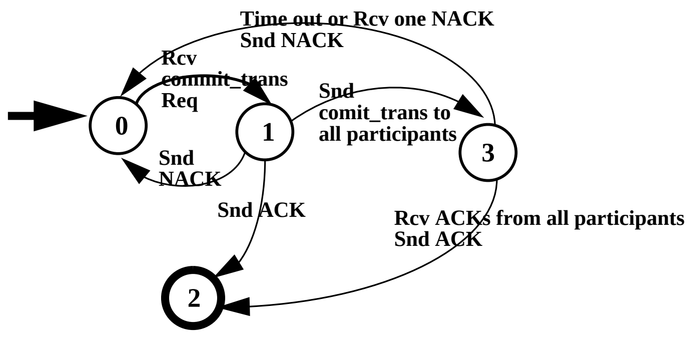
NET_COMMIT_TRANS response:

IE Type	Data Size (bytes)	Data	Mandatory or Optional	Default Value
OP_TYPE	1	NET_COMMIT_TRANS	M	
MULTIPHASE	1	RESPONSE	M	
MSG_ID	2		M	
TRANS_ID	28		M	
SM_Requester	28		M	
CG_CAUSE	1		O	OK
Send_REND	28		M	
Receive_REND	28		M	

Operation:

The operations started by receiving a net_commit_trans message are embeded in other operations. A transaction is an object that groups multiple of other requests. When one of the operation fails, the whole transaction fails and the CMs have to role back as if the transaction has never been started. Figure shows the transition starting from receiving a net_commit_trans request.

Figure 29. net_commit_trans transition diagram



0 -- initial state
1 -- net_commit_trans request pending
2 -- successful net_commit_trans
3 -- waiting for response

State transitions start at state 0, no net_commit_trans request has been received yet. When a net_commit_trans request is received, the CM goes to the state 1 (a net_commit_trans request pending). If the transaction does not exist at the current node, it sends an NACK with the cg_cause field set to TRANS_NOT_EXIST. If the current node is an terminal of the transaction, it sends ACK or Nack based on its decision. If it is not a terminal node, it will forward the net_commit_trans message to all the participants other than the one where the net_commit_trans message came from. When ACKs are received from all the participants, the CM will send an ACK to the requester. If one of the participants can not commit the transaction, the whole transaction is canceled, and the CM will role back to the state before the transaction started. In this case, an NACK is sent to the requester.

6.3.13 NET_ABORT_TRANS

Synopsis:

When a NET_ABORT_TRANS request is received, the CM role back all the actions taken for the transaction, and the CM passes the NET_ABORT_TRANS to all the neighbor CMs that are participated in the transaction. A NE-T_ABORT_TRANS request does not have to be replied.

Message Formats:

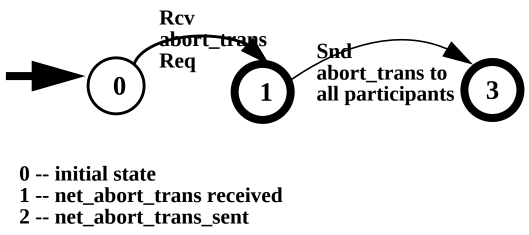
NET_ABORT_TRANS request:

IE Type	Data Size (bytes)	Data	Mandatory or Optional	Default Value
OP_TYPE	1	NET_COMMIT_TRANS	M	
MULTIPHASE	1	REQUEST	M	
MSG_ID	2		M	
TRANS_ID	28		M	
SM_Requester	28		M	
Send_REND	28		M	
Receive_REND	28		M	

Operation:

A net_abort_trans request is used to abort an ongoing transaction. Figure shows the actions the CM takes when a net_abort_trans request is received.

Figure 30. net_abort_trans transition diagram



State transitions start at state 0, no net_abort_trans request has been received yet. When a net_abort_trans request is received, the CM checks the existence of the transaction. If the transaction does exist, the CM will role back to the state just before the transaction started. If the current node is not a terminal node of the transaction, it will forward the net_abort_trans message to all the participants other than the one where the net_abort_trans message came from.

6.3.14 NET_RESET

Synopsis:

When a node reboots, all the connections are lost. It sends NET_RESET messages to all its neighbors to inform this situation. When a NET_RESET is received, the node destroys all the CGs routed through the reset node. The load of the link between the current node and the node where the NET_RESET is received is set to zero.

Message Formats:

NET_RESET request:

IE Type	Data Size (bytes)	Data	Mandatory or Optional	Default Value
OP_TYPE	1	NET_RESET	M	
MULTIPHASE	1	REQUEST	M	
MSG_ID	2		M	
TRANS_ID	28		M	
Send_REND	28		M	
Receive_REND	28		M	

Operation:

6.3.15 NET_PING

Synopsis:

A node can send NET_PING message to its neighbor to check the existence of the neighbor. When a NET_PING message is received, the node has to ACK the message. The action taken when the timer expires before the NET_PING response is received is the issue of implementation.

Message Formats:

NET_PING request:

IE Type	Data Size (bytes)	Data	Mandatory or Optional	Default Value
OP_TYPE	1	NET_PING	M	
MULTIPHASE	1	REQUEST	M	
MSG_ID	2		M	
TRANS_ID	28		M	
Send_REND	28		M	
Receive_REND	28		M	

NET_PING response:

IE Type	Data Size (bytes)	Data	Mandatory or Optional	Default Value
OP_TYPE	1	NET_PING	M	
MULTIPHASE	1	RESPONSE	M	
MSG_ID	2		M	
TRANS_ID	28		M	
Send_REND	28		M	
Receive_REND	28		M	

Operation:

7. Future Directions

7.1 Routing

7.2 Quality of Service

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.

[65] J. DeHart, M. Gaddis, and R. Bubenik. “Connection Management Access Protocol (CMAP) Specification,” Washington University, Department of Computer Science Technical Report WUCS-92-01, August 1992.

Appendix B: CMNP Operation Field Values

to be written