MULTIPOINT CONNECTION MANAGEMENT IN HIGH SPEED NETWORKS

Rick Bubenik John DeHart Mike Gaddis

Applied Research Laboratory
Department of Computer Science
Washington University
St. Louis, Missouri 63130-4899

ABSTRACT

We describe our *Connection Management Access Protocol* (CMAP) for managing multipoint connections in high-speed packet switched networks. A *multipoint connection* is a communication channel between two or more *clients* of the network, where all data sent by one client is received by all other clients. A *point-to-point connection* is a special case of a multipoint connection involving only two clients. CMAP specifies the access procedures exercised by clients to create, manipulate, and delete multipoint connections. Once a connection is established, clients exchange data using protocols that are specified separately from CMAP. We target our protocol to *Broadband Integrated Services Digital Networks* (BISDNs) employing the *Asynchronous Transfer Mode* (ATM) communication standard.

To establish a multipoint connection, a client first creates a *call* between itself and the network. The client creating a *call* is designated the *owner* of the *call*. Immediately after creation, the owner is the only *endpoint* participating in the *call*. Additional endpoints are added either by invitation from the owner, invitation from another client of the network, or by explicitly requesting to be added. These three modes are sufficient for supporting point-to-point communication (for example, a telephone *call*), many-to-many communication (for example, a conference *call*), one-to-many communication (for example, broadcast video), and many-to-one communication (for example, error logging).

1 Introduction

We present a *call* model for representing multipoint connections in a *Broadband Integrated Services Digital Network* (BISDN) [16] and present a *Connection Management Access Protocol* (CMAP) that allows network *clients* to create, manipulate, and delete *calls*. In our model, a *call* is a collection of one or more communication channels between two or more clients of the network. 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.

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

Our protocol, CMAP, defines the interface between clients and the network used to create, manipulate, and destroy *calls* in a connection oriented network. CMAP does not address the procedures for data exchange—we leave this to other protocols which are specified separately (such as TCP [7]). CMAP is layered on top of the *Asynchronous Transfer Mode* (ATM) [1] [5][15] communication standard as a *User-to-Network Interface* (UNI) protocol. The primary difference between CMAP and other connection oriented protocols is that CMAP manipulates multipoint connections as the fundamental object, whereas most other similar protocols are designed primarily for point-to-point connections.

The remainder of this paper is organized as follows. In Section 2, we give a brief overview of a BISDN network. In Section 3, we discuss the *call* model in detail. We describe CMAP and give examples of how it is used in Section 4. We present some of the implementation details in Section 5. Finally, we cover related work in Section 6 and summarize in Section 7.

2 Overview of a BISDN Network

An example BISDN network is shown in Figure 1. The network consists of *clients*, *exterior nodes* (nodes that inter-

From: *INFOCOM '91: Proceedings of the Tenth Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 59-68, April 1991.

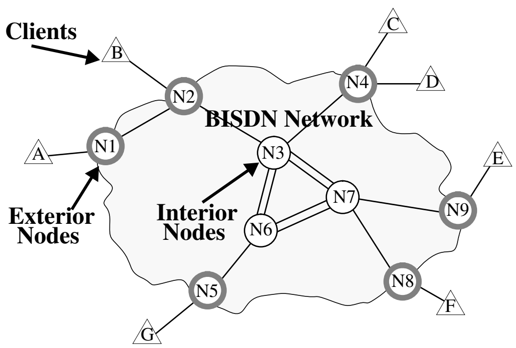

Figure 1. Example BISDN Network.

face 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 connections with other clients by sending *control* messages to exterior nodes, which carry out the requested client operations. If the request requires the services of other nodes, then control messages are exchanged between nodes (exterior and interior) in order to satisfy the request. A goal of our access protocol is to hide the complexity of the network so that it appears to the client as a single monolithic entity.

Each node in a BISDN network contains one or more ATM packet switches [8] [10] [12] [13] [14] [22] [24]. The switches route each packet to the desired destination link(s) based upon header fields in the packet. In order to keep up with line speeds (typically ranging from 100 megabits per second to over 1 gigabit per second), the switches perform all routing in hardware. Since the time interval within which each packet must be routed is very small, the switches are preconfigured with routing information during connection setup.

Figure 2 shows the architecture of Turner's *Broadcast Packet Switch* [21][22]. 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 the RN. Point-to-point cells follow an arbitrary path through the CN, 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 control cells from clients or other nodes (as distinguished by the header) to the CP via port 0.

A connection is realized in an ATM network by assigning *Virtual Path Identifiers* (VPIs) to each fiber optic link traversed by the connection [1] [5]. A VPI appears in each

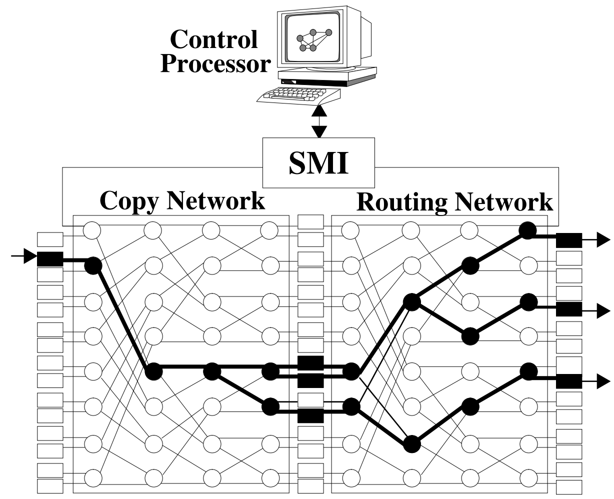

Figure 2. Architecture of Turner's Broadcast Packet Switch Demonstrating Multicast Routing.

cell's header and is used to route the cell to the desired destination. For example, consider the network of Figure 1. If a unidirectional connection is set up from node **A** to node **B** (where **A** transmits and **B** receives), the VPIs might be assigned as follows. First, link (**A**, **N1**) is assigned a VPI of 3. This VPI is used by node **A** when sending cells to the network. Next, link (**N1**, **N2**) is assigned a VPI of 6. This VPI is used by **N1** when sending cells from **A** (for this connection) to **N2**. To facilitate this, **N1** updates the tables within its packet switch to map the VPI field of incoming cells from node **A** with VPI=3 to VPI=6. Finally, link (**N2**, **B**) is assigned a VPI of 11. Node **N2** updates its tables to map incoming cells from **N2** with VPI=6 to VPI=11. This completes the connection setup. If the connection between nodes **A** and **B** is bidirectional, additional VPIs are assigned for the links going the opposite direction.†

The ATM standard refers to connections of the type in the example above as *virtual paths*. The ATM cell format also includes a *Virtual Circuit Identifier* (VCI) in the header of each cell. This field can be used to distinguish between multiple connections that all follow the same virtual path (and, thus, share the same VPIs), and to distinguish the source of transmissions in multipoint connections with multiple transmitters.

In addition to VPI/VCI connections as just described, connections can also be established using only the VCI field of the ATM header. In this mode, the VCIs are assigned to

† The same VPIs could be used for the opposite direction, but doing so requires that each node reserve the VPI for both directions, even if the connection is (or becomes) unidirectional. This prevents the VPIs from being used for other connections, reducing the total number of connections that can be supported.

each link and used for routing, instead of using VPIs. This increases the number of connections available to clients at the client interface (since the VCI field is larger than the VPI field) and also frees up the VPI for use in trunking.

While connections are necessary in wide area BISDN networks to support multicast communication and to allow packet switches to operate at link speeds, not all communication fits nicely into the connection-oriented model. Research is currently under way to add datagram service to BISDN networks [2]. We intend to support datagrams in our network through means separate from CMAP, but do not address them further in this paper.

3 The Call Model

A network client communicates with other clients by creating and manipulating *calls*. A *call* is a distributed object maintained by the network that describes the communication paths that connect clients. Initially, a *call* has a single communication channel, which we call a *connection*. The owner of a *call* (the creator) can add connections to the *call*. This provides for multiple connections within a single *call*, where, for example, one connection might be used for video and another for voice. When a *call* is first created, the owner is the only *endpoint* in the *call*. Additional endpoints may be added by: 1) invitation from the owner, where the invited party has the option of refusing the invitation, 2) request from a client not currently in the *call* to be added, where the owner has the option of denying the request, or 3) request from a third party, not necessarily in the *call*, to add a client, where both the owner and the client being added have the option to refuse. Our *call* model is *dynamic* in that the number of endpoints and number of connections in the *call* can vary over time, while communication is taking place.

The characteristics of *calls* and connections are listed in Table 1. The creator of a *call* defines the *accessibility* of the *call* to new endpoints. It may be defined as: 1) CLOSED, where no new endpoints may be added except at the invitation of the owner, 2) QUERY OWNER, where the owner has the option of refusing any attempt to add an endpoint, or 3) OPEN, where any endpoint may be added without the owner having the option to refuse. Another *call* characteristic, the *monitor flag*, controls whether the owner will be notified when an endpoint is added or deleted from a *call*.

TABLE 1. *Call* and Connection Characteristics.

Call Characteristics:	
accessibility	whether *call* is OPEN (can be freely joined), CLOSED (cannot be joined), or QUERY_OWNER (can be joined only with owner's permission)
monitor_flag	whether owner is to be notified of endpoint additions and deletions
endpoint_list	list of endpoints participating in the *call*
connection_list	list of connections comprising the *call*
Connection Characteristics:	
bandwidth	bandwidth reserved for connection
permissions	default receive and transmit permissions for new endpoints joining *call*

Endpoints within a *call* are assigned *receive/transmit permissions* for each connection in the *call*. An endpoint can have both receive and transmit permission, receive or transmit permission, or no permission. Each connection has default receive/transmit permissions that are assigned to endpoints when they join the *call*. CMAP operations allow an endpoint's permissions to be modified after it has joined. An example of where an endpoint might be assigned neither receive nor transmit permission on a connection is if the endpoint is a participant in a video conference *call* and does not have video equipment, but wishes to participate in the audio portion of the *call*, where the audio portion is carried on a separate connection. In this case, the permission for the video connection would be neither receive nor transmit, while the permission for the audio connection would be both.

Each connection in a *call* has a *bandwidth* characteristic. When adding a connection to a *call*, the owner explicitly specifies the bandwidth requirements. This information allows the network to make performance guarantees to the clients by restricting the total expected bandwidth allocated on each link to less than the link's capacity. Requests that would exceed the capacity of any link traversed by the connection are denied. The bandwidth requirement is specified as integral units of: 1) *maximum bandwidth*, 2) *average bandwidth*, and 3) *burst length*. The maximum and average bandwidth apply to the aggregate load generated by all transmitting endpoints and both measures are specified as ATM cells per second. The burst length applies to each individual transmitting endpoint and is specified as the number of ATM cells transmitted in a maximum sized burst. Several different link allocation strategies are possible using these characteristics. In our prototype network, we intend to initially use only the maximum bandwidth, but have included the other fields in the *call* model for future use. The bandwidth limits are enforced in network hardware.

4 Connection Management Access Protocol

4.1 Protocol Architecture

Clients of BISDN networks create, manipulate, and destroy *calls* by sending CMAP messages to exterior nodes of the network (Figure 3). These CMAP messages are trans-

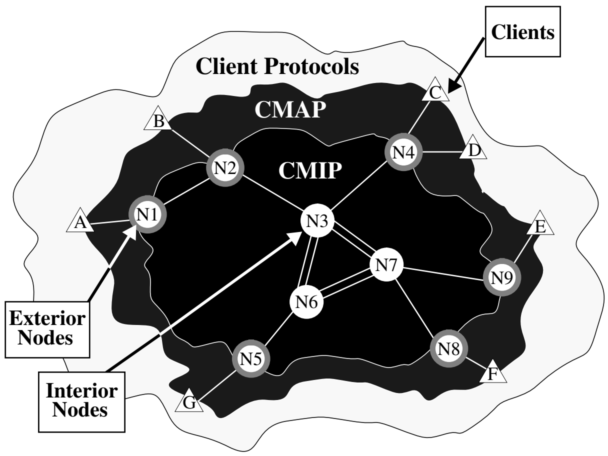

Figure 3. Layered View of the Protocol Architecture.

mitted in ATM control cells, which are distinguished from data cells by specifying the *signaling* cell type in the header. The exterior nodes communicate with one another and with interior nodes by sending *Connection Management Internal Protocol* (CMIP) messages. The CMIP messages are used to exchange, among other information, the virtual path identifiers (VPIs) and virtual channel identifiers (VCIs) used to establish connections (Section 2). CMIP messages are also sent in ATM control cells and are distinguished from CMAP messages by a type value contained in the information field of the ATM cell.

Figure 4 shows a simplified view of the client, exterior node, and interior node interfaces to the network. Client application programs (or higher level protocols) initially create connections using CMAP, then transfer data directly using the ATM protocol. CMAP and CMIP messages arriving at

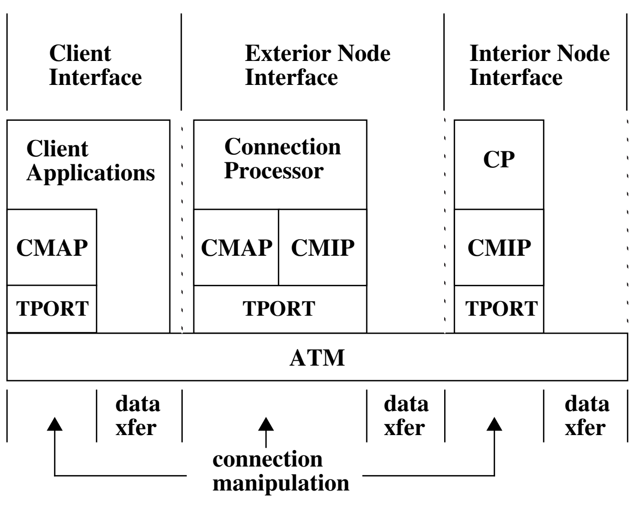

Figure 4. Protocol Interfaces to Network.

exterior nodes are routed to, and processed by, the connection processor (CP). Data cells are routed in the switch hardware and do not require CP processing. Interior nodes are identical to exterior nodes, except that CMAP processing is not required. The TPORT box shown in Figure 4 represents the transport protocol necessary to fragment, reliably transfer, and reassemble CMAP and CMIP requests using the 53 byte ATM cells. This corresponds to the functionality of the ATM *Segmentation and Reassembly* (SAR) sublayer and *Convergence Sublayer* (CS).

4.2 Addressing

In our prototype network, we identify network clients using the ISDN 15 digit international address (CCITT Recommendation E.164 [4]) concatenated to a 4 byte sub-address field (Figure 5). We split the ISDN address into four

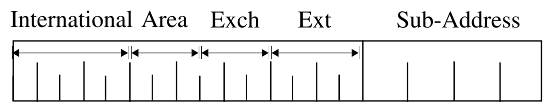

Figure 5. Address Format.

subfields: International Code (5 BCD digits), Area Code (3 BCD digits), Local Exchange (3 BCD digits), and Local Extension (4 BCD digits). This addressing scheme allows for hierarchical routing like that used in the public telephone system. We intend to modify this addressing scheme in the future to support different types of addresses so that other schemes, such as internet IP addressing [7], can also be used.

Clients are connected to the network through *client interfaces* (Figure 6). This interface is referred to as a *termination*. All terminations are uniquely labeled with *termination identifiers*, where the termination identifier corresponds to the first four fields of the address (Figure 5). CMAP only deals with termination identifiers, reserving the sub-address field for higher level protocols. Additionally,

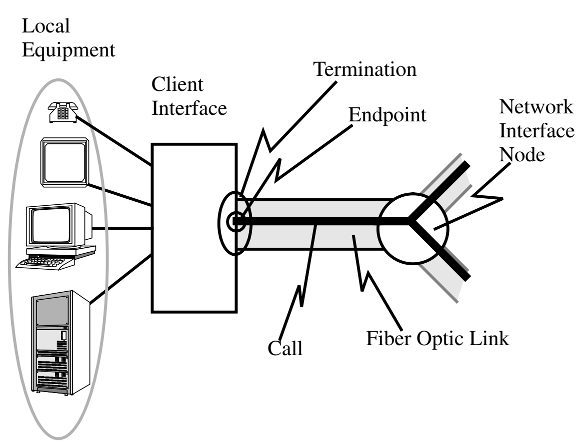

Figure 6. Client/Network Interface.

CMAP allows more than one termination identifier to be assigned to a given termination.

When a *call* is established, it is assigned a *call identifier* consisting of the termination identifier and a *local call identifier*, assigned by the client. The local *call* identifier differentiates multiple *calls* originating from a given client. The *call* identifier uniquely labels each active *call* in the network.

Each client participating in a given *call* is assigned an *endpoint identifier* by the network. If, for example, the owner of a *call*, client **X**, wants to disconnect client **Y**, **X** sends a control message to **Y**, using **Y**'s endpoint identifier to uniquely identify **Y**'s connection to this *call*.

A client can participate in many *calls* simultaneously. Additionally, a client can act as two or more clients, with respect to a given *call*, by having multiple endpoints of the *call* mapped to its interface. In this case, the client uses multiple endpoint identifiers and each endpoint is treated by the network as a separate client.

4.3 Operations

A brief description of each CMAP operation appears in Table 2. The operations are divided into four classes: *commands*, *prompts*, *notifications*, and *queries*. A *command* is an operation that a client uses to request a service from the network. The network responds to each command with an acknowledgment. A *prompt* is an operation that the network uses to request an action by a client. Clients respond to prompts with an acknowledgment, then receive a confirmation from the network when the request is complete. A *query* is used by clients and the network to request information from one another. The receiving party (network or client) responds to the query with an acknowledgment. Finally, a *notification* is used by the network to inform the client of a change in network or *call* state. No acknowledgment is required for notifications. While a command or query operation is pending (before an acknowledgment is received from the network), the client may abort the operation.

TABLE 2. CMAP Operations.

Name	Description
Commands:	
open_call	establish a *call* containing a single connection between a client and the network
close_call	terminate a *call*
add_endpoint	add a new endpoint to a *call*
drop_endpoint	delete an endpoint from a *call*
open_con	add a new connection to a *call*
close_con	remove a connection from a *call*
mod_call	modify *call* accessibility and monitor characteristics
mod_con	modify a connection's bandwidth and default receive/transmit permissions
mod_endpoint	modify an endpoint's receive/transmit permissions
client_reset	inform network of an client reset (failure)
Prompts:	
invite	invite a client to join a *call*
Queries:	
check_join	give owner option to grant or deny client's request to join *call*
Notifications:	
announce_ep	inform owner that a new endpoint has joined *call*
drop_from_call	inform an endpoint that it is being dropped from a *call*
announce_drop	inform owner that an endpoint has dropped out of a *call*
announce_change	inform an endpoint that a change (for example, in the number of connections or bandwidth) has taken place in a *call* it is participating in
network_reset	inform endpoint that a network reset has occurred

4.4 Examples

4.4.1 Point-to-point *Call*

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

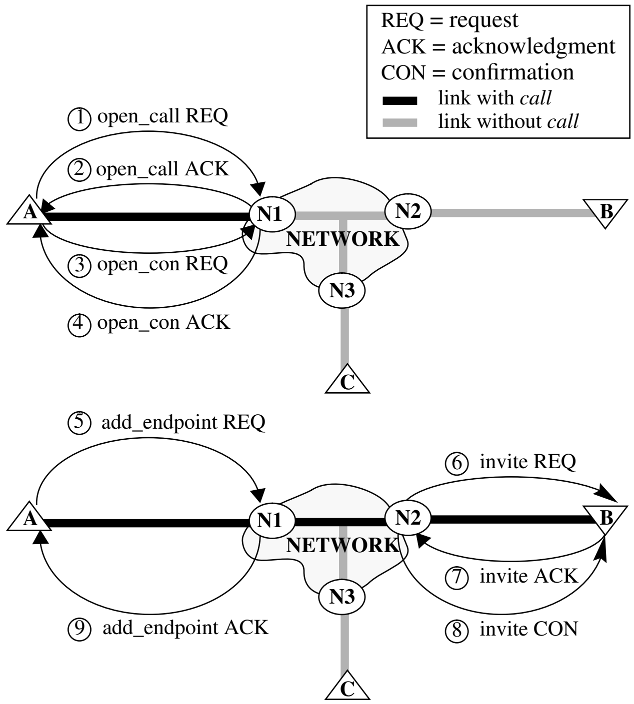

Figure 7. Building a Point-to-Point *Call*.

knowledges **A**'s **add_endpoint** request in step 9. These last two messages occur concurrently and may be received in reverse order.

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

4.4.2 Many-to-Many *Call*

The previous example is extended to a many-to-many *call* by issuing another **add_endpoint** request for client **C**. All three clients, **A**, **B**, and **C,** can issue the request, although if the accessibility is CLOSED, requests by **B** and **C** will be denied, and if the accessibility is QUERY_OWNER, **A** will be asked if the addition is acceptable before **C** is added. Figure 8 shows the steps when **C** attempts to add himself, assuming QUERY_OWNER accessibility and monitoring by **A**. In step 1, **C** sends the initial **add_endpoint** request. In steps 2 and 3, **A** receives the **check_join** request from the network and acknowledges the request (granting **C** permission to join). In steps 4 and 5, the network adds **C**, then responds to **C** with an **add_endpoint** acknowledgment and sends an **announce_ep** request to **A**. As in the point-to-point case, node **C** must be prepared to receive data immediately after issuing the **add_endpoint** request.

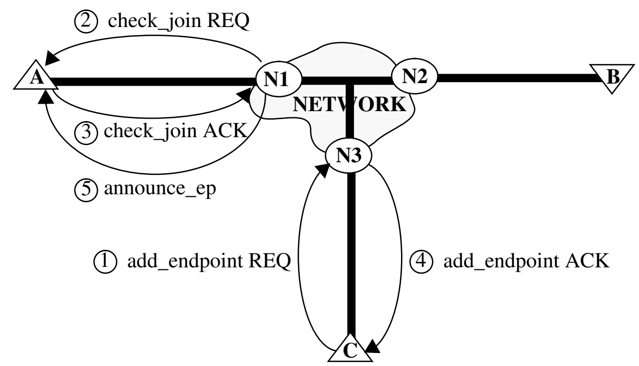

Figure 8. Building a Many-to-Many *Call*.

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

4.4.3 One-to-Many *Call*

A one-to-many *call* is created and manipulated in the same manner as a many-to-many *call*, where the permissions at *call* creation are set so that the owner is the only endpoint that transmits and all other endpoints can only receive. The main differences in the way these *call*s are handled occur within the network, where bandwidth is only allocated in the direction of the owner to the other endpoints, rather than in both directions, as with a many-to-many *call*.

If a one-to-many *call* is used to support broadcast video, the network clients would presumably issue **add_endpoint** requests when they wish to join the *call* (as in the previous example, Section 4.4.2). If the broadcast source is pay-for-view, the owner could create the *call* with QUERY_OWNER accessibility. Otherwise, the *call* might be created with OPEN accessibility.

5 Implementation

The information pertaining to each *call* is distributed throughout the network. No one node has complete information about a *call*. Each node maintains, for each *call*, the accessibility, monitoring status, and a pointer to the owner of the *call* (Section 3). For each connection, each node maintains the bandwidth, default receive/transmit permissions, the outgoing links down which receivers exist, and the incoming links down which transmitters exist. Since endpoints can participate in a *call* without receiving or transmitting on some connections, data for a connection does not necessarily flow through all nodes and, therefore, over all links (Figure 9). However, the direction down which trans-

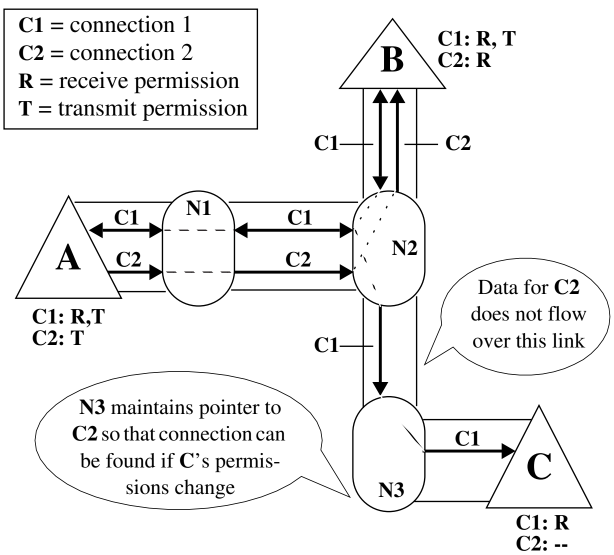

Figure 9. Multipoint *Call* With Two Connections, C1 and C2, Where C2 Does Not Flow To Node C.

mitters and receivers can be found is always stored in case a new endpoint joins the *call*, with receive and/or transmit permissions on a given connection, at a node through which data for that connection is not flowing.

The interior representation of a *call* is that of a tree, where the owner can be thought of as the root. As new endpoints are added, the tree branches out to these endpoints. Cycles and directed acyclic graphs (DAGs) are prevented in *calls* since they could lead to infinite looping of messages and duplicate messages.

5.1 Connection Management Internal Protocol

As mentioned in Section 4.1, the *Connection Management Internal Protocol* (CMIP) is used by nodes within the network to manipulate connections. Each CMAP operation spawns one or more CMIP operations to carry out the request. We are currently in the processes of specifying CMIP. To briefly illustrate some of the operations, consider the following example (Figure 10). Assume that the owner of a *call* issues a CMAP **add_endpoint** request (step 1). Furthermore, assume that the *call* is OPEN so that the owner does not have to be queried when a new endpoint joins. The following CMIP operations result:

- First (step 2), a CMIP **net_invite** request is sent by the owner's interface node (**N1**) to the new endpoint's interface node (**N4**), generating a CMAP **invite** request at the new node (step 3).
- Next, the new endpoint acknowledges the **invite** (step 4), triggering CMIP **net_join** requests that are sent back towards the owner (steps 5 and 6). Each node on the route reserves the necessary bandwidth for the *call*. When a node currently participating in the *call* is reached (**N2**), the **net_join** propagation stops.‡

‡ In this example, we assume all connections of the *call* needed by **C** have data flowing through **N2**. If this were not the case, additional CMIP messages would be required to bring these connections to **N2**.

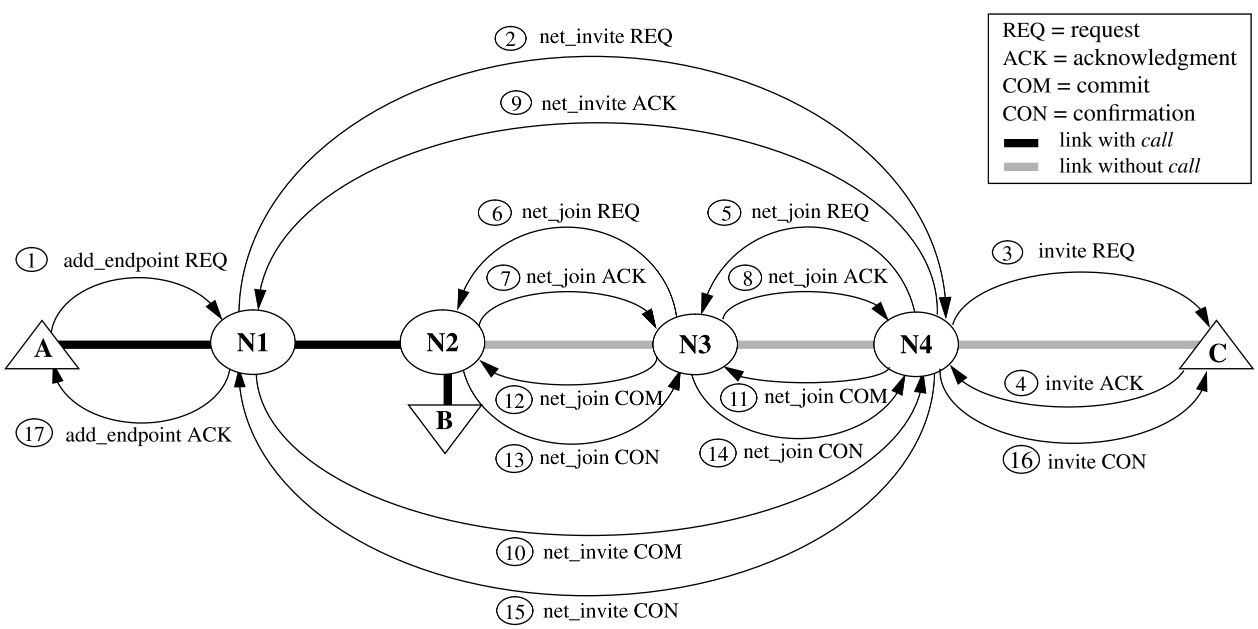

Figure 10. CMIP Messages Generated For add_endpoint Request.

- **Net_join** acknowledgments are then sent back towards the new endpoint (steps 7 and 8). When the new endpoint's interface node receives this message, it sends a **net_invite** acknowledgment back to the owner's interface node (step 9).
- The owner's interface node then sends a **net_invite** commit message back to the new endpoint's interface node (step 10), which sends **net_join** commit messages to the nodes that previously received the **net_join** requests (steps 11 and 12). As each node receives the commit message, it performs the necessary update in the switch to support the new connection(s).
- Finally, **net_join** confirmation messages are sent back to the new endpoint's interface node (steps 13 and 14), then a **net_invite** confirmation message is sent back to the owner's interface node (step 15). These messages spawn the CMAP **invite** confirmation (step 16) and **add_endpoint** acknowledgment (step 17) messages.

As this example illustrates, both the **net_invite** and **net_join** operations have two phases, each requiring two messages. In fact, in the current design, all CMIP operations have these two phases. In the first phase, a request is sent and an acknowledgment is received. This tells the originator of the request that the necessary resources have been reserved and that the nodes participating in the request are prepared to make the desired updates. In the second phase, a commit is sent and a confirmation is received. The commit message causes the participating nodes to carry out the update, then respond when done. Upon receiving the confirmation message, the originator of the request knows that all of the required updates have completed.

5.2 Network Transactions

The two phases used by CMIP operations are analogous to the two phase commit protocol used in distributed databases [3][18]. The network is essentially a distributed database and CMIP operations are atomic transactions on that database. The transactions can be *nested* [17], as the **net_invite** and **net_join** requests were above. All of the problems associated with transaction processing in a distributed database are also present in network updates. For example, two CMIP operations can conflict (such as two **net_join** requests on the same *call*) and, therefore, these operations must be serializable. Also, failures can occur at any stage of a CMIP operation and the proper steps for recovery must be taken. In designing CMIP, we are using database theory as a guideline for ensuring correctness.

5.3 *Call* Construction

The CMIP example in Section 5.1 illustrates an important aspect about the way in which new endpoints are added to a *call*. Namely, new endpoints are added by building the connection from the new endpoint to the *call*. The alternative would be to build from the *call* to the new endpoint. The latter approach leads to problems if, when building towards the new endpoint, a node participating in the *call* is reached. In this case, a connection has been established that breaks from the *call* then rejoins the *call*, thus creating a DAG. Since the *call* must remain a tree for correct operation, the new branch needs to be destroyed. Building in the opposite direction avoids this problem.

5.4 Routing

Routing information is used to locate clients when adding endpoints to a *call* and when building connections from new endpoints to the existing *call*. We rely on a *routing server* located at each node to decide which link should be traversed in these cases. The protocol used by the routing servers to exchange routing information is specified separately from CMAP and is, along with the routing algorithm, currently under development.

6 Related Work

Our work extends earlier work on connection management done at Washington University by Haserodt and Turner [11]. In particular, our *call* model and CMAP operations are an evolution of this earlier work.

Minzer and Spears [16] discuss multipoint connections in BISDN networks. They describe many of the services that BISDN networks may eventually be used for and outline an abstract *call* model that supports many of these uses. Their model is very preliminary, however, in that many aspects of the model are left unresolved and no mention is made of how *calls* are created, manipulated, and destroyed. In contrast, we present a concrete *call* model, along with a protocol and specific operations that can be performed on *calls* to support the same types of services as those discussed by Minzer and Spears. One difference between their view of a network and our view is that they believe higher level information processing functions, such as code conversion, should be performed within the network, whereas we feel the network should provide a flexible interface so that these functions can be easily provided as client-based services (where certain clients offer high level services to other clients). Our approach simplifies network hardware and software functionality, while in no way compromising the services available to clients.

The TCP protocol [7] is similar to CMAP in that both protocols pertain to connection oriented communication. However, TCP specifies both connection setup and data transfer procedures, whereas CMAP is only concerned with connection setup, leaving data transfer to other protocols. In fact, TCP could be layered on top of CMAP to provide a data transfer capability. One capability lacking in TCP is CMAP's support for multipoint connections. Another difference is CMAP's support for bandwidth guarantees, not available in TCP.

The X.21 [9] and X.25 [19] connection oriented protocols are also similar to CMAP. Like TCP, they specify procedures for data exchange in addition to circuit setup. Multipoint connections were proposed, in CCITT Recommendation X.2, as an optional service feature for X.21 and X.25.* Unlike CMAP, however, these multipoint connections are *static* in the sense that all participants are specified when the connection is created. CMAP supports a *dynamic call* model, where participants can be added and deleted at any time during the *call's* existence. This dynamic model is necessary, for example, in order to support flexible broadcast communication, where viewers can tune in and out of ongoing *calls*. Another difference between CMAP and the X.21 and X.25 protocols is that CMAP allows multiple connections to be grouped into a single *call*, whereas X.21 and X.25 only support a single connection within a *call*.

The XTP protocol [20] [23] supports one-to-many unidirectional multicast connections, similar to CMAP. However, connection creation is considerably different. In XTP, a sender creates a one-to-many connection by multicasting an initiation message.†† All potential receivers are assumed to be listening for the multicast request so that they can initialize their internal state in response to the initiation message. There is apparently no way, within XTP, to guarantee that all receivers get this message. In contrast, CMAP requires that each receiver be added separately (instead of with a single multicast operation), and CMAP guarantees that all receivers have been connected when acknowledgments are returned. Once a connection has been created in XTP, receivers have the ability to dynamically join and drop out of the ongoing connection. But, this must be done by the receiver and cannot be initiated by the sender or any other client (as in CMAP). XTP addresses both connection setup and data transfer procedures (including procedures for reliable one-to-many communication). As with TCP, these capabilities of XTP could be layered on top of CMAP.

7 Summary

We described a *call* model for representing multipoint connections in a BISDN network and a protocol, CMAP, for creating, manipulating, and destroying *calls*. Our model and protocol support: 1) point-to-point communications involving two clients, such as that used in simple telephone calls and data exchange, 2) many-to-many communications involving from a few to hundreds of clients, such as that used in teleconferencing and in multicast group communication [6], 3) one-to-many communications involving, perhaps, tens of thousands of clients, such as that used in broadcast video distribution, and 4) many-to-one communications, involving an arbitrary number of clients, such as that used in error logging. Once a *call* has been established, separate protocols facilitate the transfer of data from one endpoint to one or more other endpoints.

We are currently working on the design and implementation of CMIP, the connection management internal protocol, as well as the associated network transaction mechanism and the routing protocols.

* It is unclear whether any network ever implemented this optional multipoint connection feature.

†† In XTP parlance, a "FIRST" message is sent.

References

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

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

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

[4] 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.

[5] CCITT. *Recommendations Drafted by Working Party XVIII/8 (General B-ISDN Aspects) to be Approved in 1990,* Study Group XVIII—Report R 34, June 1990.

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

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

[8] 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.

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

[10] 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.

[11] 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.

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

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

[14] 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.

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

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

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

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

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

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

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

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

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

[24] 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.