AN ARCHITECTURE FOR CONNECTION MANAGEMENT IN A BROADCAST PACKET NETWORK

Kurt Haserodt and Jonathan S. Turner

WUCS-87-3

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130-4899

Abstract

Connection management refers to the collection of protocols, algorithms and data structures used to establish and maintain multipoint connections in a broadcast packet network. This report documents an initial proposal for a set of connection management protocols that allows users to specify arbitrary multipoint connections in a flexible and application-independent fashion. It also contains a set of detailed scenarios illustrating how the protocols would be used.

This work supported by the National Science Foundation (grant DCI 8600947), Bell Communications Research, Italtel SIT and NEC.

An Architecture for Connection Management in a Broadcast Packet Network

Kurt Haserodt
Jonathan S. Turner

DRAFT

1. Introduction

The following is a description of a connection management scheme for the Broadcast Packet Network (BPN) described in [4]. This packet-switched network supports services ranging from conventional telephone calls through broadcast video. Though the BPN hardware uses packet switching at the data link level, the software transmission protocols support logical connections as their major network level mechanism for communication.† This necessitates the design of a scheme for the creation, management, and destruction of these circuits.

A justification for connection-based protocols in general data networks is given in [2]. The node processing speed required by the enormous bandwidth involved in the BPN make datagram protocols impractical. Less burden is placed on the network if the load is limited by connection denial rather than by datagram discarding. The routing overhead associated with datagrams is severe even with point-to-point connections, but it increases greatly with multipoint connections required for broadcast video. Additionally, connection based protocols are consistent with the predominant uses of the BPN (voice and video communication) which are themselves connection oriented.

Connection management in the BPN refers to a collection of algorithms for creating, maintaining, and destroying large, multi-point connections. In the case of the BPN, the connection concept must include point-to-point, broadcast, and conference connections as specific examples. The connections can, therefore, have an arbitrary number of endpoints.

In [3] it is shown that endpoints of a connection can have more than one intervening switch, so there is no obvious, central point of control. Connection management algorithms, therefore, must be distributed. Also for a large, reliable, network, keeping global information is undesirable. The algorithms must operate quickly even without such information.

This document is a proposal rather than a description. No actual connection management scheme for the BPN has been implemented nor simulated up through the time of its preparation. Our hope is that, by making a detailed proposal, many of the design problems, and further research issues will surface.

This paper is divided into several sections. Section two describes some scenarios we feel are indicative of the use of the BPN. In section three, the BPN is presented as abstractions at

---

† The alternative here would be datagrams. The BPN does support datagrams, but the predominant use should be through logical connections (virtual circuits).

three different levels of the detail described in [3] and [4]. Sections four and five then present the connection management algorithms at for the lower two levels described, while section six explores applications at the higher level. Section seven includes a description of further research.

2. Usage Scenarios

Usage scenarios are given in this section. Each scenario represents some typical use of the BPN and has requirements which are different from those of the other scenarios. Together, and in hybrid combinations, the requirements should exercise all facilities of the BPN. As a protocol design's features are presented later, its solution to a particular problem posed by one of these scenarios is also given.

The scenario requirements fall into four broad categories:

- Connection topology--point-to-point (telephone call), multi-point (conference call), or broadcast (television) connections are possible.
- Connection initiation--some scenarios require that a customer connect with another customer (telephone call) while others require that a customer connect to a broadcast channel (television).
- Bandwidth requirements--this can vary in degree (voice versus video) and can be of similar amounts in both directions of a connection (telephone) or can be vastly different (television with no upstream capability).
- Data reliability--loss or alteration of data can be tolerated in some cases (voice) and not in others (file transfer).

Connection topology and initiation features are stressed in the example scenarios presented in this paper. Bandwidth requirements are handled to a lesser extent, and data reliability is not addressed. In this vein, of the scenarios presented below, the simple telephone call, broadcast television, and video lecture seem to address most of the connection topology and initiation features the we deem are a novel feature of this network. The other scenarios are interesting as applications on top of the network and will be used as examples in the section concerning that level.

2.1. Simple telephone call

A simple telephone call represents the most basic, traditional use of the BPN. Any design of a new network must encompass this at the very least. Such a call is a voice-only, duplex, point-to-point connection with one end-point initiating the call. While these calls will represent a small amount of the total bandwidth transmitted by the BPN, most connections are likely to be of this type.

2.2. Point-to-point file transfer

A file transfer is similar to the telephone call. It differs in that it will have initial setup requirements (e.g. protocol negotiation) and, while it will require duplex operation for most protocols, the sending bandwidth requirements of the file origination point should be much higher than the receiving bandwidth. The bandwidth required will typically be much higher

than that needed for a telephone call, but the duration of the call should be shorter. Also, this type of service's bandwidth requirements can vary. While the customer might prefer a higher rate of data transfer, they might accept a lower bandwidth as opposed to none.

Note that this application is the basic operation for "electronic mail transfer." It is also indicative of any service which does not have a person at one of the endpoints, and so might not require real time response.

2.3. Voice-only conference calls

Conference calls provide the first exercise for the broadcast capability of the BPN. Data generated by each user must be transmitted to all parties involved in the call. Customers should also be able to originate such calls themselves, and contact others to join the call. Other users should be able to initiate a join to a call already in progress. The only area in which this would differ from a video conference call would be that of bandwidth requirements.

2.4. Broadcast television

Broadcast television involves one source transmitting data to several sites. These sites request connection to the broadcast channel. The bandwidth towards the receivers should be very high, while that returning to the broadcaster will be low or nonexistent. Also the number of receiving sites is probably quite high and their geographic extent quite wide.

2.5. Lectures

Lectures are like broadcast television in that there is one central broadcaster (the lecturer) and several receivers. There should be some bandwidth required on an upstream audio path however. Also the scale of a lecture should not be as large, and the sending site may wish to initiate contact. Again, the only difference between this and a video conference would be that the bandwidth required for the upstream information would be lower.

2.6. Electric meter reading

Electric meter reading would typically involve a request from the electric utility to the meter and then a reply from the meter. A logical connection is probably wasteful here, and this scenario should help to justify the utility of datagrams, and why they are used in the BPN.

3. BPN Abstractions

It is helpful to view the BPN on different levels, each one more detailed than the previous. We see several levels of abstraction, but this paper deals with three major ones: the customer premises equipment (CPE) level; the BPN level; and the packet switch (PS) level. The BPN level is the first where the network itself is seen, and so is termed "level N," while the CPE level is "level N+1," and the PS level is "level N−1."

Other levels above and below the three described here can be seen. At level N−2, different types of PSs are identified (network interfaces (NIs) and packet switches). Also each PS (or NI) is decomposed into the switch modules. At level N−3, the switch modules are further decomposed into packet processors, connection processors, and the switching fabric. At level N−4, the switching fabric is decomposed. With these lower levels, communication between each

of the components described in [4] can be specified. A detailed description of component interaction at level N–4 can be found in [1].

Also, certain components at level N+1 can be decomposed. We see the CPE as being composed of a customer premises interface (CPI) and other devices which require use of the network to transmit data (e.g. a telephone). With this in mind, a level N+2 can be seen. Communication between customer-supplied devices using the network could be specified here.

While it is within the scope of the design of the BPN that we specify, fully the connection management algorithms and communication protocols necessary at levels N and below, it is only necessary to partially specify these for levels above N. This partial specification allows certain basic services (e.g. voice communication) and serves as an example for further development.

3.1. Overview of the CPE Level

On the CPE level, communication is seen as shown in Figure 3.1. The CPE can transmit data over several *logical circuits* each going to one or more destinations. Each circuit can be composed of a few *channels*, each with differing bandwidth properties. At this level the connection is already established. The CPE is only communicating with another CPE. Connection management here involves only the devices which constitute the CPE. For the CPE to initiate a connection to a new CPE it must use the facilities provided by the CPE/network interface.

Figure 3.1 shows a single channel, bidirectional, multipoint call between CPEs A, C, and D. Also, a dual channel, point-to-point connection exists between CPEs A and B. A CPE can be an endpoint of more than one connection, and it uses the logical circuit number (LCN) to distinguish the connection. A sends data to B using LCN y. It uses LCN x to broadcast data to both C and D simultaneously. This connection is typical of a conference call.

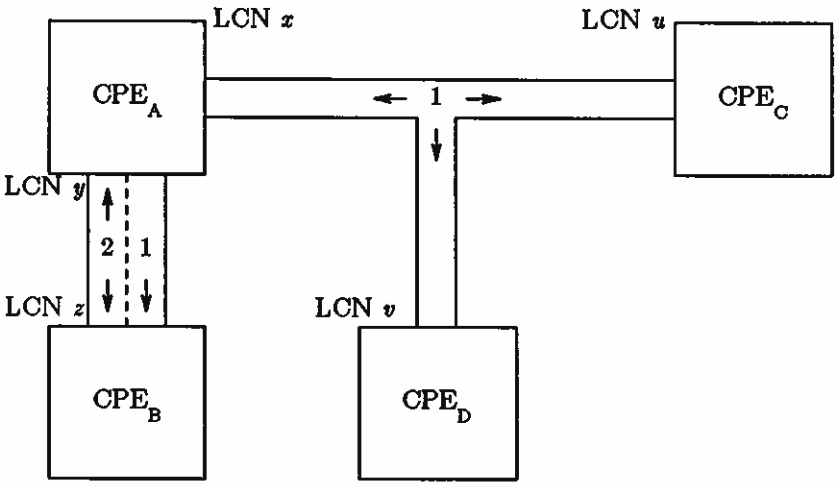
Channels further divide the connection. In Figure 3.1 the circuit between A and B contains two channels. A and B can both send and receive on channel 2, but channel 1 can only be used by A to send to B. This type of connection might be found in a hybrid scenario, with A and B communicating via a voice channel (2), and A sending video images to B as an adjunct to their conversation.

3.2. Overview of the BPN Level

The network is first seen on the BPN level. It can be viewed as a single intermediary in CPE to CPE communication (see Figure 3.2). On this level the services provided by the network and the protocol used to provide these services is specified. Each CPE must make requests of the network to build, modify, and destroy the logical circuits it requires for communication with other CPEs. The information transmitted can be identified as data or control. The network intercepts and interprets the control information, making the required changes to the circuit (connection)‡. The data is passed on to the destination CPE.

---

‡ The terms *circuit* and *connection* are synonymous. *Circuit* is usually used when data information is being discussed, and at the CPE level while *connection* is used at the BPN and lower levels when control information is the topic.



LCN - logical circuit number
CPE - customer premises equipment

Figure 3.1: Level N+1 Abstraction of BPN (CPE level)

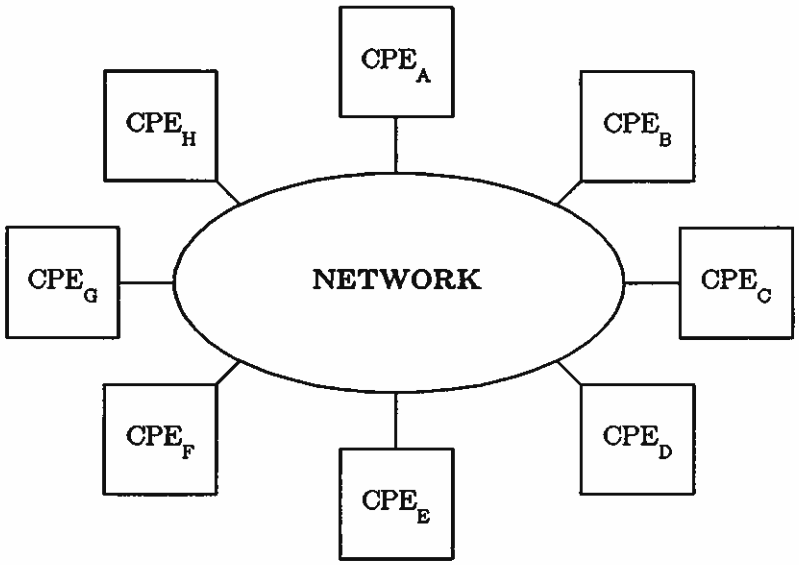


Figure 3.2: Level N Abstraction of BPN (BPN level)

3.3. Overview of the PS Level

The BPN is actually not one central switch, but a set of distributed, interconnected packet switches. At the PS level, information flows from the CPE to a PS, and is then routed, possibly through other PSs, to the destination CPE (see Figure 3.3). Some connection information must be maintained in each PS, but it is neither necessary nor desirable for each, or one central, PS to maintain a copy of all the required information. A mechanism for inter-PS communication is also required at this level.

The general format of the two following sections is the same. A specification of the data structures which describe the basic entities at that level are first given. The operations allowable on these structures are then described. And finally, a specification of how these operations are combined to perform the actions dictated by certain scenarios is given.

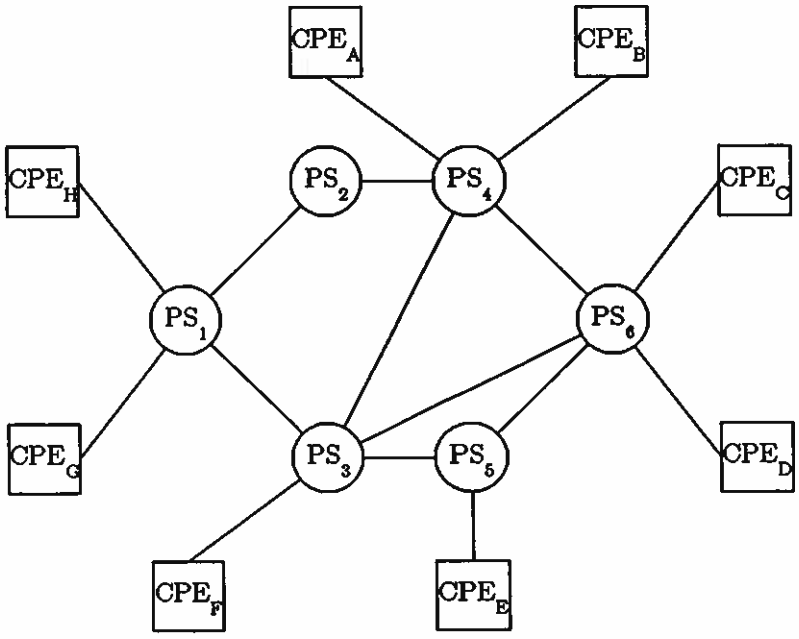


Figure 3.3: Level N–1 Abstraction of BPN (PS level)

4. Level N Specification: Broadcast Packet Network (BPN)

While in the scenarios we have described several distinct connection types (e.g. point-to-point, broadcast, and conference), we prefer to have a single, more general connection concept that includes each of these as special cases. This simplifies both the operations required for connection manipulation, and the protocol necessary to perform them. These operations are chiefly concerned with connection creation, growth, and termination. There are two ways that connections can grow: users within the connection can request that new endpoints be added and users outside the connection can request to be added on (this is needed for large broadcast connections).

Of the three types of connections considered above, the conference connection is the most general. In this type of connection, packets sent by any participant are forwarded by the network to all others. By adding suitable control mechanisms, this type of connection can be tailored to other uses. The notion of channels is one such mechanism. A channel is characterized by its average bandwidth, the set of endpoints that can transmit on the channel and the set that can receive. To implement a broadcast connection, we use one channel on which the only transmitter is the broadcast source, while all other endpoints can only receive. To add a low bandwidth upstream capability, we provide a second channel on which the broadcast source receives and all the others transmit. The concept of a connection containing multiple channels is very flexible and allows bandwidth to be allocated in the network only where it is needed.

4.1. Connection Structure

A *connection* is an abstract data type. CPEs operate on connections by sending control packets to the network. Each control packet results in an operation on the connection's data and a response packet from the network, so this interaction can be viewed as a procedure call. The network specifies what *procedures* (command packet types) it will allow.

Before the connection structure itself can be described, a description of group of ancillary data structures is needed. In the following, declarations similar to what you would find in the C programming language are given. A `BYTE` is an 8-bit value, and a `WORD` is a 16-bit value. These should give an indication of the data value magnitudes and structure sizes involved, though they might change in actual implementations.

Logical circuit number (*lcn*) `WORD`

The logical circuit number identifies the circuit on which communication between the network, and the CPE is occurring. One physical line between the CPE and the network may contain several logical circuits. Also two CPEs may be communicating with each other over different logical circuits. The network performs the translation.

Address (*adr*) `BYTE [6]`

The address should uniquely identify any CPE which is connected directly to the network. A further refinement of the address is possible (e.g. having the first byte specify a certain geographic area) and in many cases desirable, but is not necessary for our purposes.

Connection identifier (*cid*) `{ adr, lcn }`

The connection identifier is simply the combination of the address of a CPE (usually the originator of the connection), and the logical circuit number on which that CPE and the network communicate. This uniquely identifies any connection, because the address is unique, and only one connection is allowed per logical circuit.

Bandwidth (*b*) { WORD, WORD, WORD }

The customer specified required average bandwidth must be kept for each channel. Additionally, some indication of the expected burstiness is needed. Three parameters should be sufficient for such a specification if the mechanism suggested in [5] is used.

Channel permission (*p*) (NO | TRANSMIT | RECEIVE | TRANSRECV)

To implement single direction broadcast connections, and private conversations inside of conference calls, channels have permissions associated with them, on a per-CPE basis. A CPE might have no access to a channel, it might be allowed to only transmit or receive on the channel, or it might be allowed to do both.

Channel (*chan*) { *b, p* }

A channel can now be specified. It has a bandwidth specification, and a default permission. Since the number of channels allowed in a given connection will be small, identification of the channel is usually by its position in a list. Note also that all CPEs using a channel have the same bandwidth restrictions, but they may have different permissions (see below). The channel identifier is short, because there only a few channels. The ability to enforce bandwidth restrictions must be built into the network hardware, and each channel must have a distinct means of controlling the bandwidth used.

Endpoint (*endp*) { *adr, lcn, *p* }

The connection has a list of endpoints associated with it. Each endpoint has a distinct address and logical circuit number. And each endpoint has its own set of channel permissions (the * indicates a list), so, for example, one endpoint might be able to transmit and not receive on a given channel while another might be able to receive only. This would exemplify a typical broadcast connection. The channel permissions are ordered by channel number. Note that this specification does not preclude the possibility of one CPE being two endpoints (different *lcns*, so the terms are not actually synonymous, though we tend to think of them as such).

Accessibility (*J*) (ANY | NONE | CHECK)

The accessibility of a connection dictates the ability for other CPEs to join to it. ANY and NONE means any endpoint or no endpoint respectively is allowed to join, while CHECK means that a designated endpoint of the connection must first be consulted.

Connection { *cid, J, endp, *endp, *chan* }

The connection data structure is that which is maintained by the network. Associated with each connection is a connection identifier which is the address and lcn used by the creator of the connection. The accessibility of the connection is stored. The creator of the connection is designated as the root. It is also the only endpoint which can modify the connection's other parameters, and it is the one which authorizes connections if the accessibility specifies that that is necessary. A list of endpoints which are part of the connection is kept. Finally, an ordered list of channels, which are part of the connection, is given.

Figure 4.1 shows a connection with endpoints. In this example, the top box gives the connection identifier (A45) which also designates CPE A as the root. The fact that no outside CPE can join to the connection is indicated by the accessibility of N. The connection has two channels. The first has a bandwidth of 50M (megabits/second), and a default permission of receive only (R). The second has a bandwidth of 50K (kilobits/second), and a default permission of transmit and receive (TR). The connection also has two endpoints. The first is the root (CPE A) which is using LCN 45, and has transmit permission (T) on channel one. The second is CPE B which is using LCN 8. By

the omission of channel two permissions for either endpoint and the channel one permission for CPE B, it is assumed that they have the default permissions (TR for channel two and R for channel one). This type of figure will be used in describing the scenarios in this section, and in this case might exemplify a video broadcaster (A) and receiver (B) with a two-way voice channel between them.

4.2. Connection Operations

All connection operations are performed by the network. They can be categorized into four major types: root to network, network to endpoint, endpoint to network, and network to root. Operations directed to the network might also come from CPEs which are not part of a connection, and so are not, technically, endpoints. They are noted when given. Operations are implemented by sending and receiving control packets between the CPE and network. Control packets are not associated with any logical circuit, so they are intercepted by the appropriate agent inside the network or CPE and interpreted rather than being passed on as data. CPE to CPE control packets are passed as data over the logical circuit. Every operation specified below initiates a reply to the sender. The information in the reply is also loosely specified.

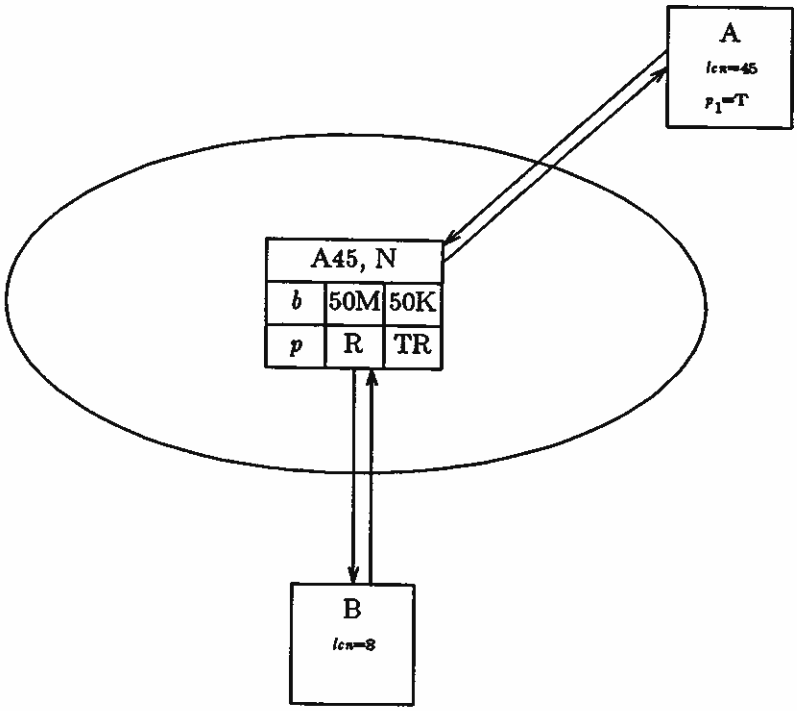


Figure 4.1: A Sample Connection at the BPN Level

4.2.1. Root to network

These are creation, control, and termination operations for a connection. Only the root can modify connection parameters or create channels on a connection. Note that the root is also an endpoint, so it can execute any of the endpoint to network operations specified later, though only one (`get_info`) makes any sense in this context.

Open a connection `open_con` (*adr, lcn*)
Open a new connection within the network. The CPE allocates an unused logical circuit, and then asks the network to perform the open. A connection structure will be allocated, with the *adr* and *lcn* forming the connection identifier. The connection will have one endpoint (the initiating CPE) which is also the root. A default accessibility, and set of channels will be attributed to the connection. This default would be that which is required by the most common network service‡. The reply from the network includes a status code and the connection identifier. This *cid* is used in subsequent commands.

Close a connection `close_con` (*cid*)
The root is the only endpoint which can close the connection. Doing so causes all resources for the connection to be deallocated. The network responds with the status of the operation.

Open a channel `open_chan` (*cid, b, p*)
The root also has the ability to create new channels associated with the connection. The bandwidth and default permissions are specified at this time. The reply includes the status and channel number. While the CPE allocates logical circuits (LCNs), the network is responsible for allocating channels. This command will fail if the network cannot supply the bandwidth requested.

Close a channel `close_chan` (*cid, chan*)
The specified channel is closed, and any resources allocated to it are freed. Individual endpoint's permissions on this channel would be changed to that of no access. A reply with status is returned to the root.

Modify a channel `mod_ch` (*cid, chan, b, p*)
The channel's bandwidth and default permission are changed. The current permissions of the endpoints are not changed. A reply message with status is returned. The request will fail if there is not enough available bandwidth to handle the specified increase.

Add an endpoint `add_ep` (*cid, adr*)
An endpoint structure is allocated with the default permissions. An `invite` message (see below) is then generated and sent to the CPE whose address is *adr*. The reply does not return from the network until the `invite` completes. It includes the result of the `invite`.

Delete an endpoint `del_ep` (*endp*)
The endpoint structure associated with the endpoint is deleted, and no more data will be sent to it. Also, the endpoint will not be allowed to send data through the connection. The reply includes the status of the deletion.

---

‡ We envision this to be a connection allowing no outside access and consisting of one 50K TR channel. This should be sufficient for a simple voice telephone call.

Modify connection permissions `mod_per` (*cid, endp*, $p_1, \ldots, p_m$)
The permissions for each of the connection channels for the specified endpoint will be changed to the values given. A mechanism to change only certain permissions should be provided. A reply with the status of the change is returned.

Modify connection accessibility `mod_acc` (*cid, J*)
The accessibility of the connection is changed to *J*. A reply message with status is returned.

4.2.2. Network to endpoint

Operations of this type are usually the result of some request made by the root (e.g. an `add_ep` operation to the network triggers an `invite` to an endpoint). Other, internally triggered, messages might occur.

Invite to join connection `invite` (*cid, J, m*, $b_1, \ldots, b_m, p_1, \ldots, p_m$)
The network sends this to a CPE as the result of an `add_ep` operation. It contains the full description of the connection as it currently exists. The CPE should allocate a logical circuit, and send this back to the network in a reply message. Otherwise it should send back a negative reply with some reason for not accepting the invitation. The network will then send back the result to the root. Upon success, further negotiation between the CPEs can then occur along the established data path.

4.2.3. Endpoint to network

These operations are only useful when the CPE does not have any association with the connection. Note that the root to network operations specified above are also endpoint to network operations (if the endpoint is the root).

Join a connection `join_con` (*cid, lcn*)
When a CPE wishes to attach to a conference call or a broadcast it allocates a logical circuit and issues a `join_con` request to the network. The identifier for the requested connection is provided. If the accessibility of the connection allows anyone to join or no one to join then a reply packet with the appropriate status is immediately returned. If, however, it is CHECK, a `join_req` message (see below) is sent to the root, which will then decide. The reply message then from the `join_con` will be that the root gives to the `join_req`.

Disconnect from a connection `dropout` (*cid*)
When a CPE wishes to disconnect from a connection it issues a `dropout` request to the network. The network then deallocates any resources required by the CPE's association with the connection. The CPE should have communicated with other CPEs in the connection to let them know that it was dropping out. The reply message returns a status code.

Get connection information `get_info` (*cid*)
A CPE might require certain information from the network about a connection, such as the addresses of those CPEs involved in the connection. This information might be needed to help decide whether a `join_con` request should be issued. This request for information will be granted only if the accessibility of the connection allows anyone to join. A reply message is returned with the status of the request plus any granted information. It is likely that this request will have parameters which will allow the CPE

to specify the information desired.

4.2.4. Network to root

These are special cases of network to endpoint messages. They might be generated in response to a request by another endpoint or internally by the network.

Request for join to connection `join_req(`*endp*`)`
This is generated by the network when an endpoint requests to join a connection and the accessibility is CHECK. The root will decide whether to grant the request and send a reply message indicating its decision. This will trigger the reply to the endpoint asking for access.

4.2.5. Combinations

The operations just presented are at a basic level. Due to the fixed packet size dictated for the BPN in [4], it would be inefficient to require a CPE to use an entire packet for one message, so we intend to allow message and reply combinations. For example, the creation of a non-standard channel might require several modifications after the initial open. We would allow these modifications to be combined into one operation (from the CPE's standpoint), and also replies to them as one operation.

Combinations are not used in the examples given below. They are optional and merely serve to allow more efficient operation of a physical implementation.

4.2.6. Completeness

For completeness, there should be a corresponding network to endpoint message for every root to network message (except for `open_con`, where there are no other endpoints). For example, if the root closes the connection, each endpoint should be notified of that fact. Also such network to endpoint messages would be useful when the network needs to notify the endpoints of a connection that it is decreasing its bandwidth limit, in order to decrease congestion.

We feel, however, that these types of messages are not required at this level if a data path already exists between the root and the endpoint. A higher level CPE to CPE protocol is used to notify all the endpoints of the impending closing of the connection, and a timeout mechanism in the CPE will catch any closings about which the CPE has not been notified. Since the underlying packet transmission mechanism does not guarantee reliable delivery, the higher level protocol must implement this feature in any case. The network would indicate to the root that it is using too much bandwidth, either directly with a control message, or indirectly by discarding packets. The root would then lighten its load and tell the other endpoints to do the same. Note that the indirect method can be used with endpoints other than the root.

These same arguments apply for leaving out corresponding network to root messages for endpoint to network actions.

4.3. Scenarios

This section describes three scenarios: a simple telephone call, broadcast television, and a video lecture with upstream voice capability. For each, a diagram for the steady state is given

(i.e. after the connections are established when the data is being transferred). Then an explanation of the operations which occurred to achieve that state is given. Finally the operations required to tear down the connection are explained.

4.3.1. Telephone call

A simple telephone call has an originator and a destination. Figure 4.2 shows such a call at steady state (i.e. the CPEs are passing voice data as part of the call). The figure shows that the call is identified by A3 which indicates that the root (A) is communicating over logical circuit three. The root is also the originator of the call. This is a private call, so there is no outside access allowed which is designated by the N next to the connection identifier. This connection has only one communication channel which has a bandwidth of 50K and allows, by default, both parties to transmit and receive.

The phone call starts with the originator (A in this case) opening a connection by passing an open_con message to the network (see Figure 4.3). The connection structure is allocated, default parameters are assigned, and A is marked as the root of the connection. The network then sends a reply to the open_con apprising A of the connection identifier.

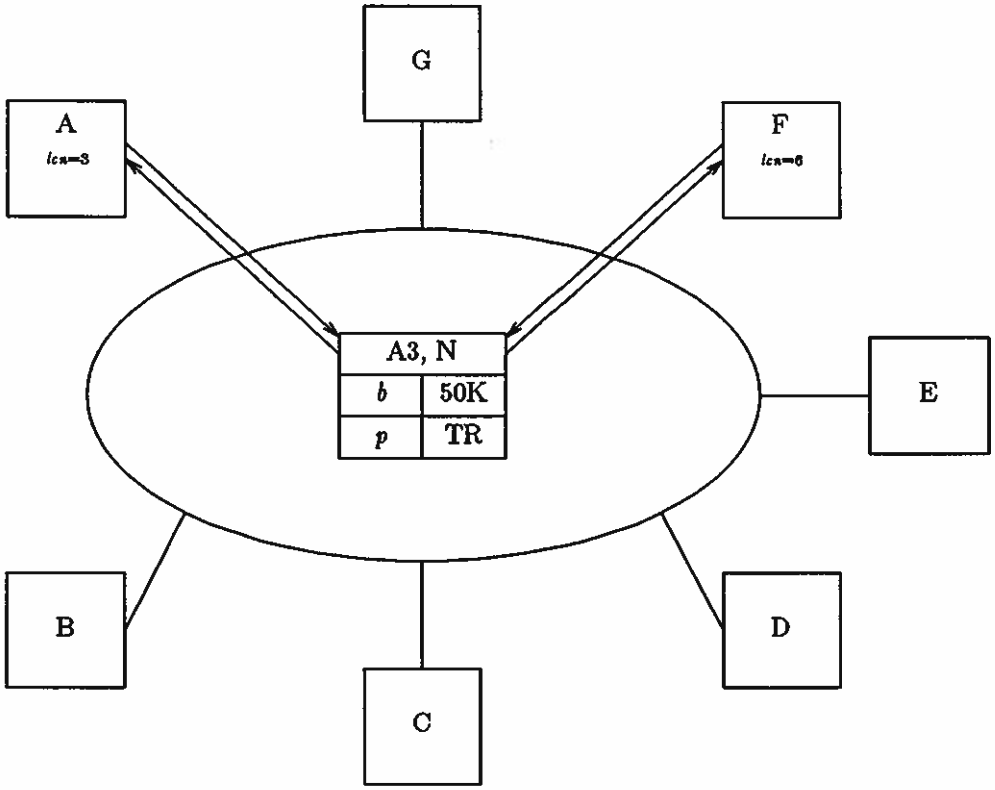
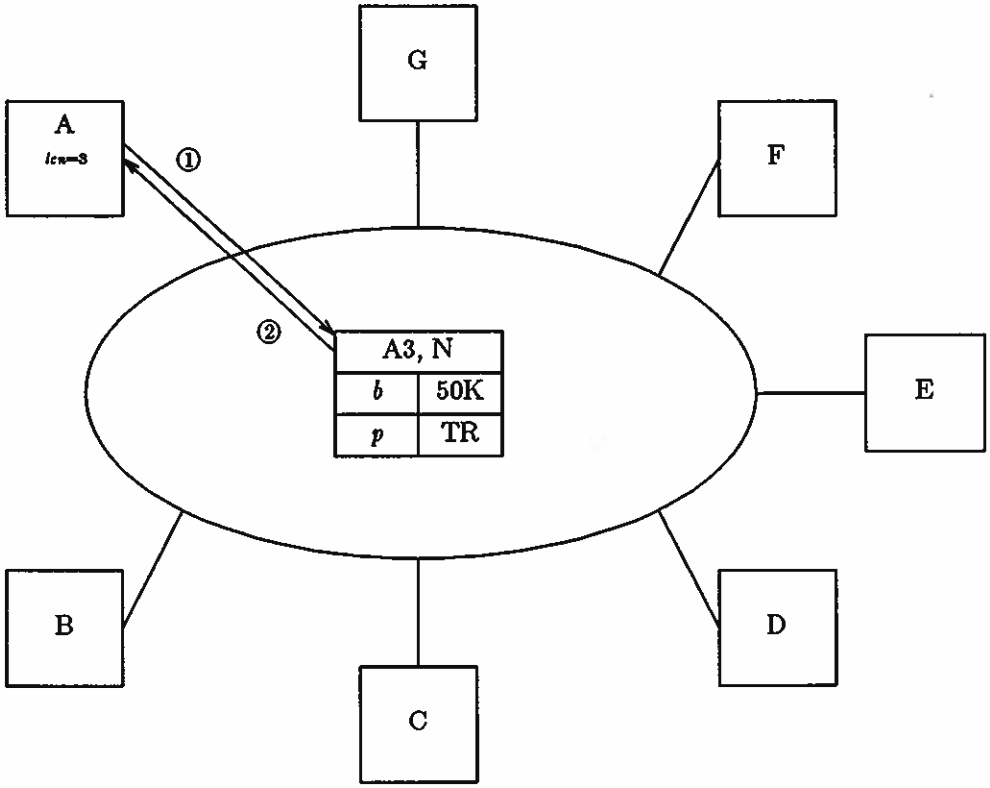


Figure 4.2: A Telephone Call at Steady State



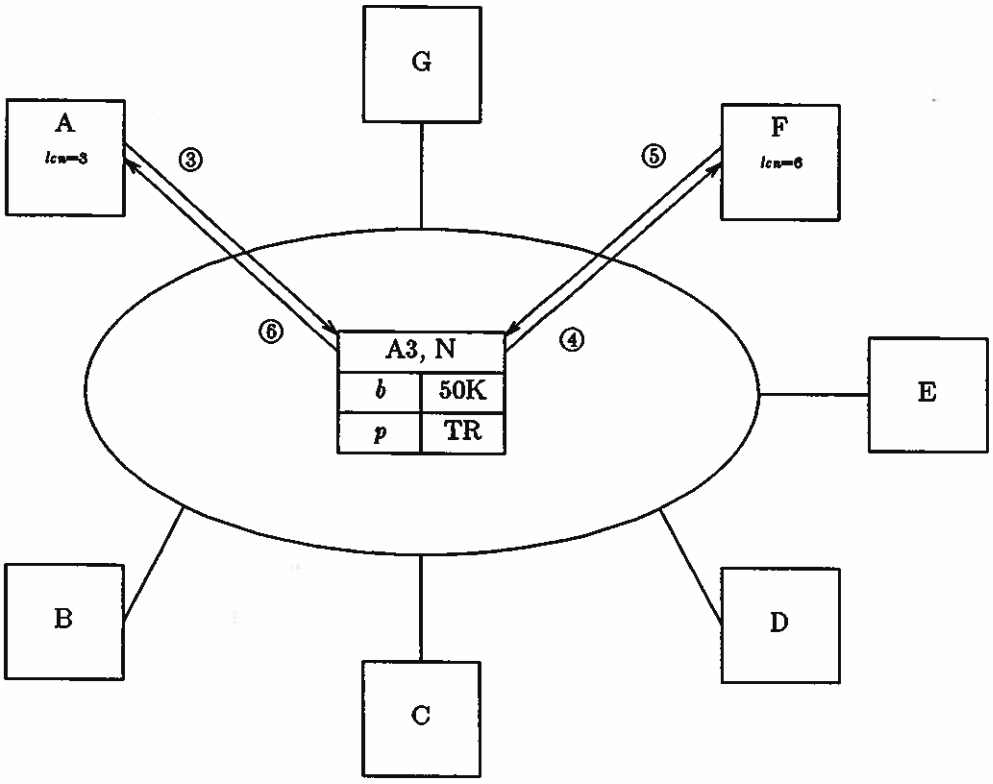
① `open_con(A, 3)`

② `r_open_con(OK, A3)`

Figure 4.3: Connection Creation for a Telephone Call

A then asks the network to add an endpoint, F, with the `add_ep` message (see Figure 4.4). This causes the network to contact F and offer an invitation to join a connection with the `invite` message. F then responds, in this case positively, by passing back status information and the logical circuit on which it wishes to communicate. The network then responds to A's initial request with the status of the invitation to F. The CPEs are now connected, and can negotiate, at a higher level protocol, such issues as which station at the location should be contacted, how it should be rung, etc.

Data is passed between the two parties involved in the call. At some point one decides to terminate the call. It notifies the other CPE, and both CPEs deallocate the logical circuit which they were using. The root then closes the connection with a `close_con(A3)`. The network deallocates the connection resources and responds to the root's request with a `r_close_con(OK, A3)` status message if all went well.



③	`add_ep(A3,F)`	⑤	`r_invite(OK,F6)`
④	`invite(A3,N,1,50K,TR)`	⑥	`r_add_ep(OK,A3,F6)`

Figure 4.4: Contact of Destination in a Telephone Call

4.3.2. Television broadcast

With a television broadcast, there is one broadcaster and many receivers. Figure 4.5 shows such a broadcast when data only, and no connection information, is being transmitted through the network. In this case the root (G) is the broadcaster, using logical circuit seven, and all the other CPEs are receivers, using the indicated LCN. Anyone can join (tune into) this broadcast. There is one 50M channel which allows reception only by default. The broadcaster has altered its permission to allow transmission only.

For television the scenario is a bit more complex. The default connection is no longer appropriate, so it must be modified. The broadcaster (G) opens the connection (see Figure 4.6), and immediately modifies the channel to reflect the bandwidth required for video, and the default permission. All endpoints will just receive data, and not transmit it. The broadcaster then modifies its own channel permission so that it may transmit on the connection. It also modifies the accessibility of the connection so that anybody can join without any other

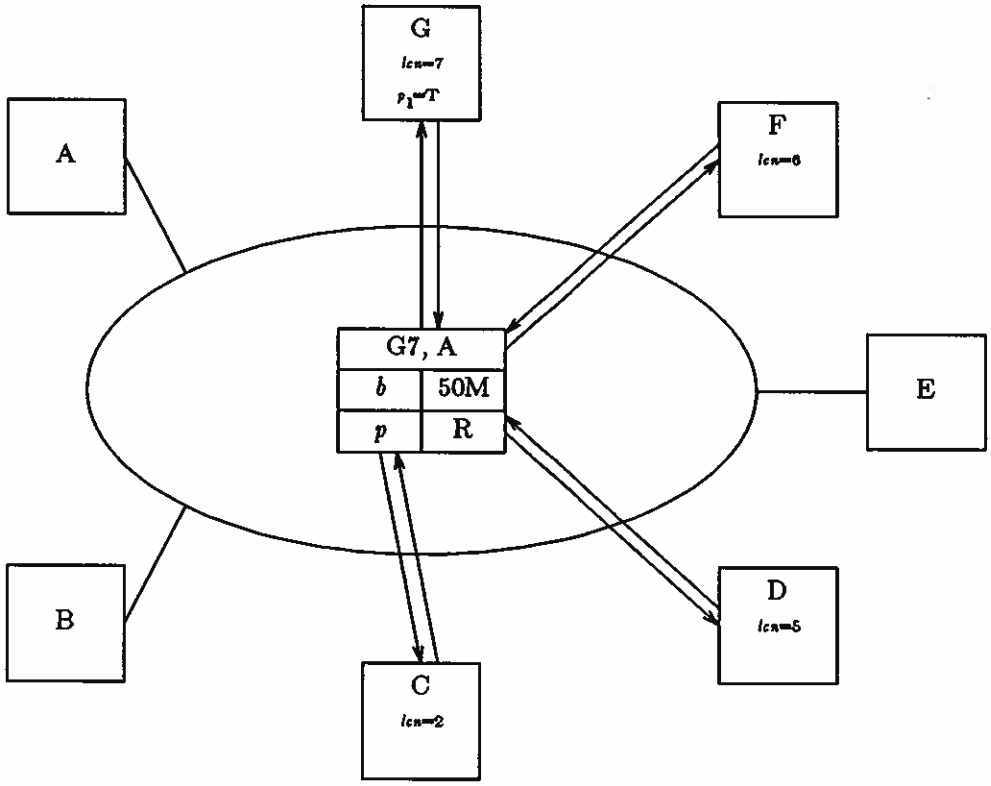


Figure 4.5: Broadcast Television at Steady State

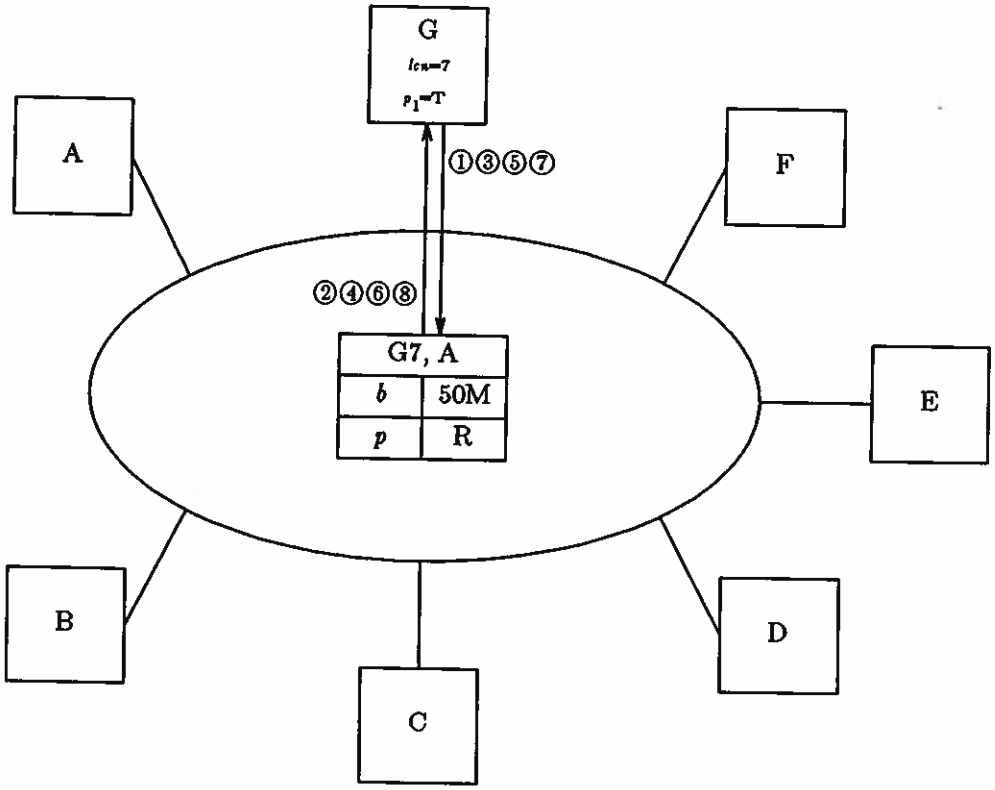
authorization taking place. After this, the broadcaster starts transmitting data which the network would promptly discard, because there is no one to receive it.

At some time, a customer will choose to connect to the broadcast. The customer (C in Figure 4.7) sends a `join_con` request to the network with the connection identifier of the broadcast to which they would like to connect.† The network sees that that connection will allow access to anyone, so it immediately connects the customer to the broadcast, and returns a status message. Other customers may attach to the broadcast in this fashion.

When the customer want to disconnect from the broadcast, it merely issues a `dropout` request to the network. The network then removes it from the list of endpoints, and returns a status message. This is shown in Figure 4.8.

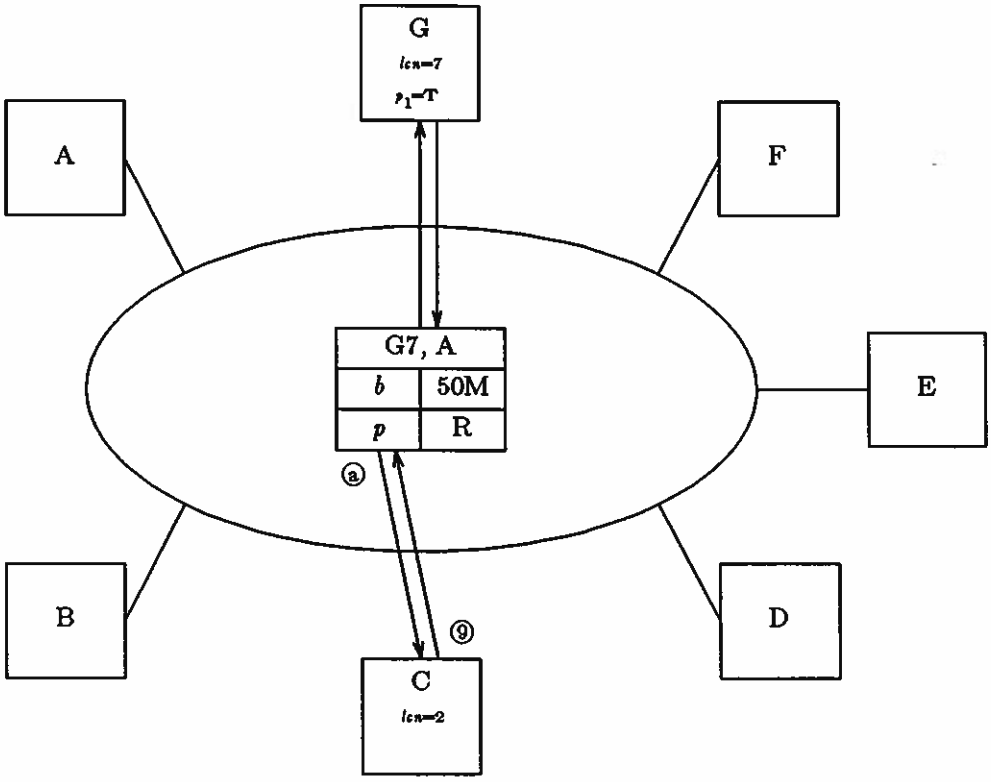
---

† The *cids* for a broadcaster should be published in some sort of a viewing guide, so that customers know the appropriate connection identifier.



①	`open_con (G, 7)`	⑤	`mod_per (G7, G7, T)`
②	`r_open_con (OK, G7)`	⑥	`r_mod_per (OK, G7)`
③	`mod_ch (G7, 1, 50M, R)`	⑦	`mod_acc (G7, A)`
④	`r_mod_ch (OK, G7)`	⑧	`r_mod_acc (OK, G7)`

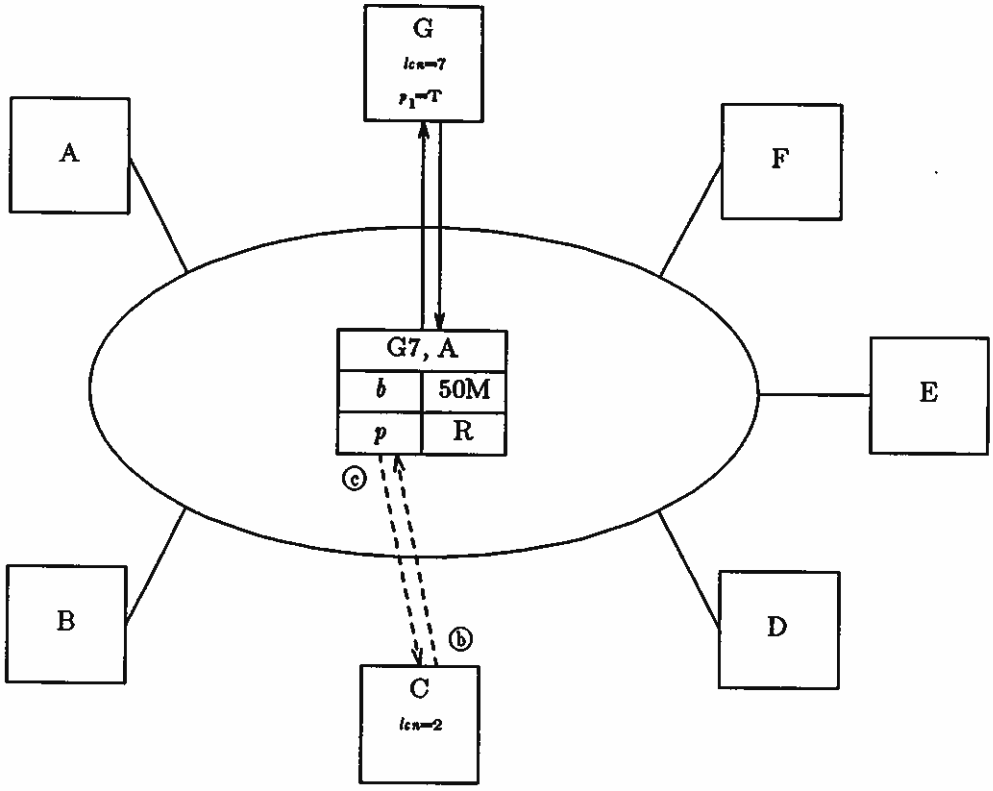
Figure 4.6: Opening a Broadcast Television Connection



⑨ `join_con(G7,2)`

ⓐ `r_join_con(OK,G7)`

Figure 4.7: CPE Requesting to Connect to a Television Broadcast



ⓑ `dropout(G7)`

ⓒ `r_dropout(OK,G7)`

Figure 4.8: CPE Requesting to Disconnect from a Television Broadcast

4.3.3. Video lecture with upstream audio

In a video lecture, there is one video (with audio) lecturer and several audience members who might like to ask questions. Their questions should be heard by the rest of the audience as well as the lecturer. Figure 4.9 shows such an event. The root CPE (B) is the lecturer. Only CPEs which have its approval (designated by the C accessibility) can join the lecture. The video channel has a bandwidth of 30M and everyone who does join can receive this, by default, but not transmit on it. The lecturer can transmit on this channel. The audio channel has a bandwidth of 50K, and the audience members can both transmit and receive on this channel. The lecturer will only receive on this channel, using the audio portion of the video channel to transmit the answers to the questions.

A video lecture begins with the lecturer (B) initializing the connection (see Figure 4.10). The connection is first opened, and the default channel parameters are modified for video transmission. Another channel for audio is required. Note that new channels have default parameters making them suitable for voice. The lecturer (root) then modifies its own channel permissions so that it can transmit only on the video channel and receive only on the audio

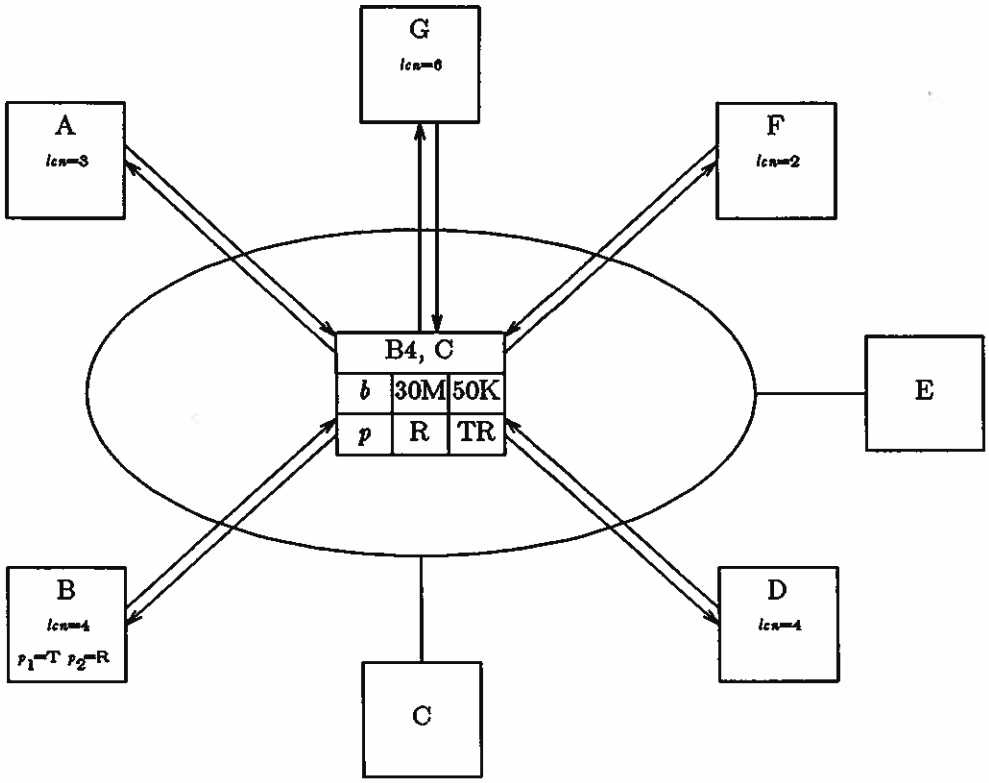
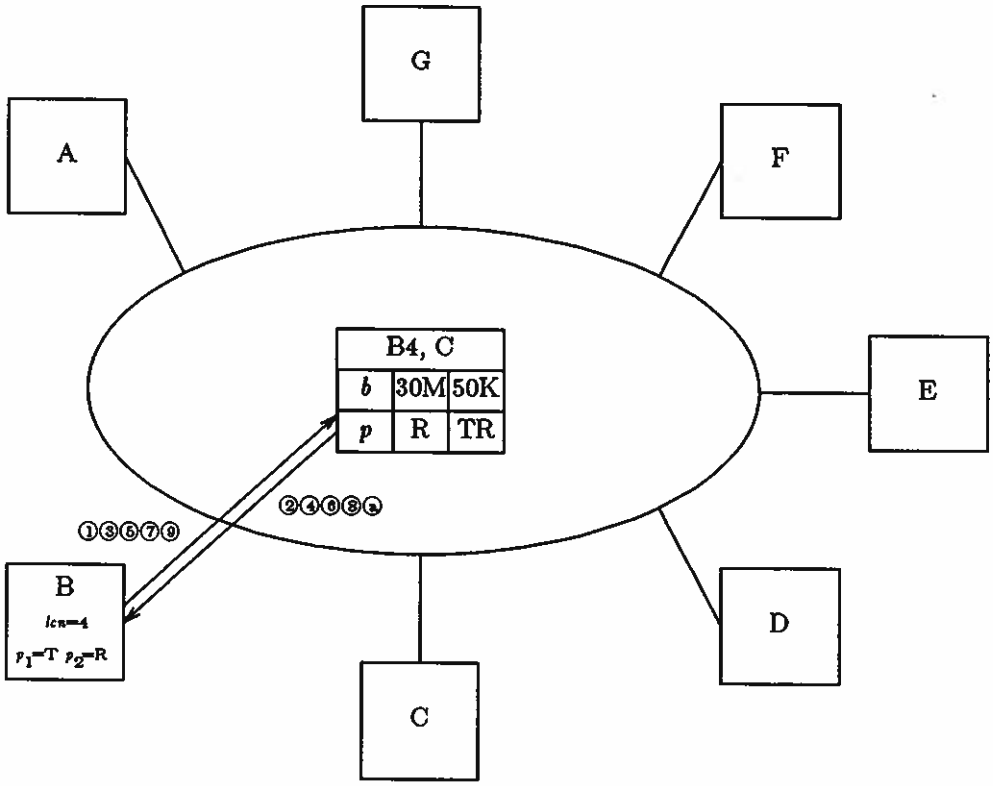


Figure 4.9: A Video Lecture at Steady State

channel. Finally, the accessibility is modified to require approval of the lecturer before a customer can join the lecture. Note that the same state could be reached by applying other combinations of connection operations.

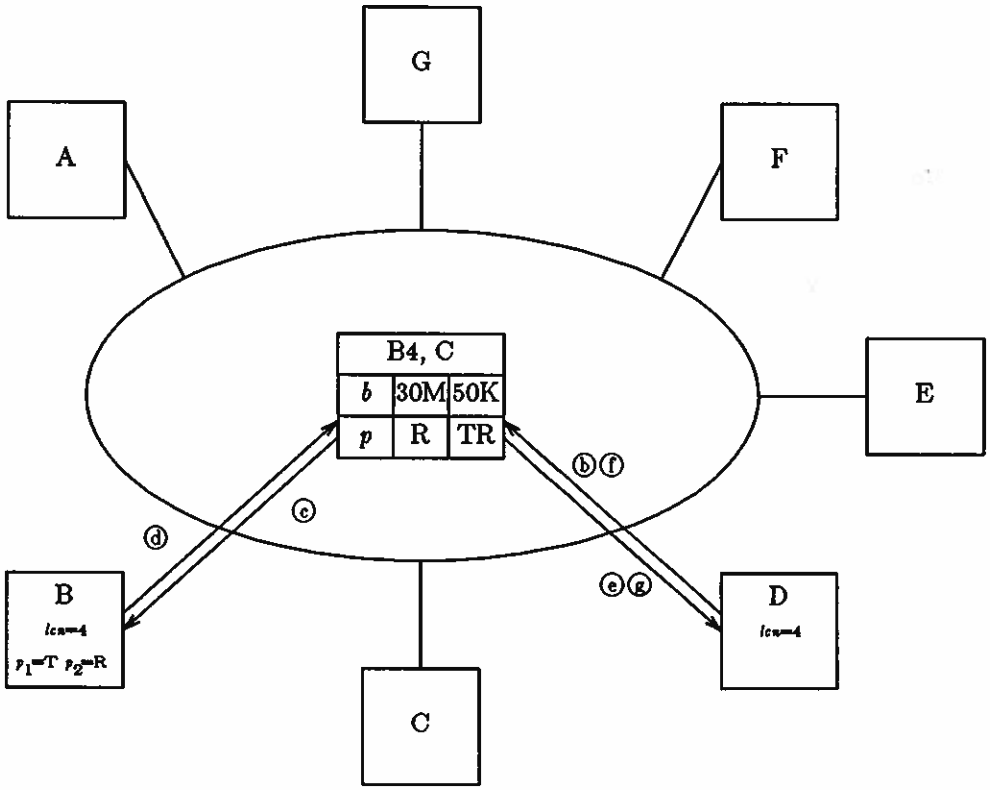
Next, D decides to "attend" the lecture (see Figure 4.11). It issues a `join_con` request, giving the logical circuit on which it would like to receive this lecture. Since the accessibility of the connection is C, a request to the root must be generated by the network. The root answers this request affirmatively, and that is passed back to D. D's channel permissions are set to the default. D then issues a `get_info` request to determine the parameters of the connection (most importantly its channel characteristics). D could not have gotten these characteristics without first joining the connection, because of the connection's accessibility.

After a while it is decided to have F join the lecture. The root (B) then sends an `add_ep` message to the network which, like in a telephone call, transmits an `invite` message to F (see Figure 4.12). F then responds with an acceptance or refusal of the invitation, giving the logical circuit it wishes to use if it accepts. The result is passed back to the root. Note that F might not have to issue a `get_info` request, because some connection parameters are passed in the `invite` message.



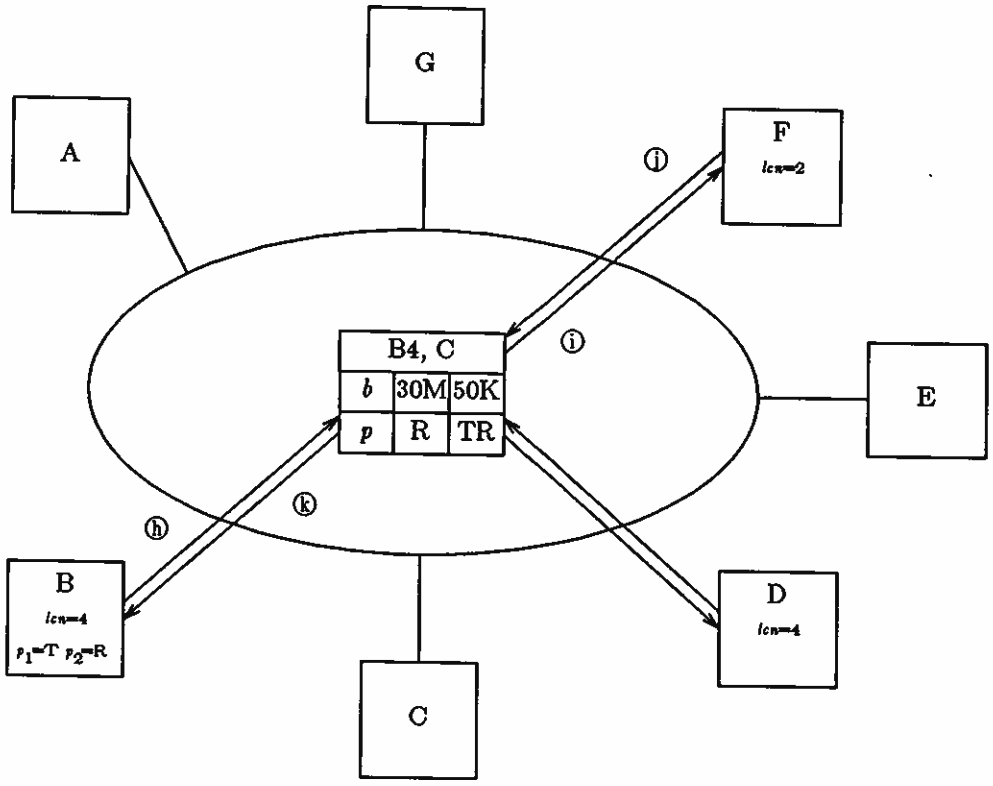
①	open_con (B, 4)	⑥	r_open_chan (OK, B4, 2)
②	r_open_con (OK, B4)	⑦	mod_per (B4, B4, T, R)
③	mod_ch (B4, 1, 30M, R)	⑧	r_mod_per (OK, B4)
④	r_mod_ch (OK, B4)	⑨	mod_acc (B4, C)
⑤	open_chan (B4, 50K, TR)	ⓐ	r_mod_acc (OK, B4)

Figure 4.10: Opening a Video Lecture Connection



ⓑ	`join_con(B4,4)`	ⓔ	`r_join_con(OK,B4)`
ⓒ	`join_req(B4,D4)`	ⓕ	`get_info(B4)`
ⓓ	`r_join_req(OK,B4)`	ⓖ	`r_get_info(OK,B4,2,30M,R,50K,TR)`

Figure 4.11: CPE Requesting to Join a Video Lecture



ⓗ `add_ep(B4,F)`

ⓘ `invite(B4,C,2,30M,R,50K,TR)`

ⓙ `r_invite(OK,F2)`

ⓚ `r_add_ep(OK,B4,F2)`

Figure 4.12: Requesting that CPE Join a Video Lecture

5. Level N–1 Specification: Packet Switch (PS)

The PS level is one below the BPN level, and so its specification requires more detail. Now the connection data must be distributed over many packet switches instead of just one network entity. Operational messages must also be routed through the network in such a way that replies can be returned. Fortunately the structures and operations required are similar to those specified at the BPN level.

5.1. Connection Structure

Like the connection at level N, the PS at level N–1 can be viewed as an abstract data type. The data for the connection is actually distributed over all the PSs involved in the call. Each of the operations at level N has an analogue at this level. The data in a PS is modified as a result of a message received, and then it may pass a message along to other PSs involved in the call so that they may modify their connection data.

PS data is as shown in Figure 5.1. The connection data is shown in dashed boxes. This data is quite similar to that required for a connection at level N. It is, however, mostly replicated in every PS through which the call passes. The links, address, and routing data are shown in solid boxes. For connection purposes they are not modified, but rather are modified as a result of other control operations.

In the following description, many of the basic data types defined in the level N section are used. The reader is referred back to that section for details.

Device Type (*dtype*) (PS | CPE)
This is used to identify the type of the device at the other end of a physical link. PS indicates a packet switch, and CPE indicates customer premises equipment. Certain messages are altered when sent to a CPE instead of another PS.

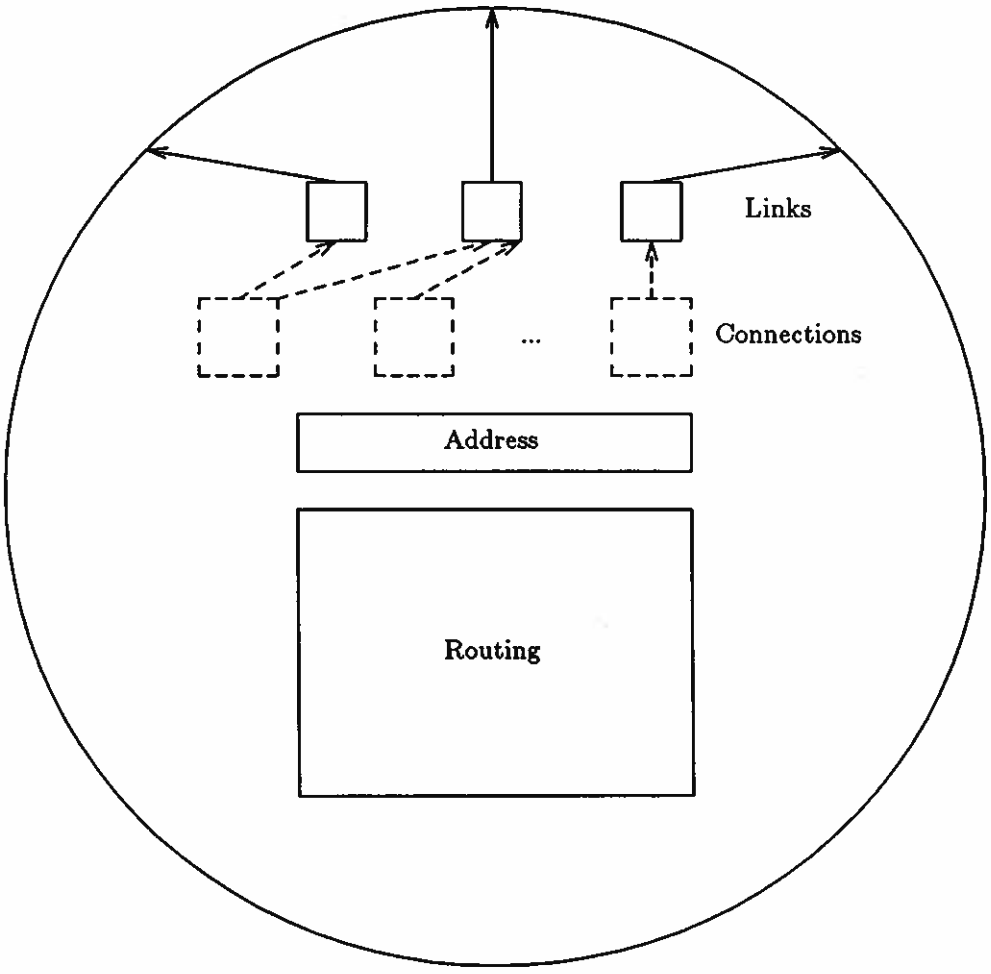


Figure 5.1: Packet Switch Data Structure

Physical Link (*plink*) { *adr, dtype, b, b* }

The physical links from a PS to neighboring PSs and CPEs are denoted by these structures. The neighbor's address, device type, and bandwidth data are given. The bandwidth specified is the total available and the total remaining. They are used to determine the ability to form a new connection to the neighboring device. Except for the remaining bandwidth, most of the data in these structures remains constant, and is only changed when the network topology changes. These are represented by the solid boxes at the top of Figure 5.1, and the associated solid arrows represent physical connections to neighboring devices.

State (*state*) (TMP | EST | CLOSING)

Links, connection structures or channels are in one of these three states at any given moment. Items in the TMP state are used to form a return path for reply messages. Items in the CLOSING state are used to mark the connection as closed, so that any other attempted operation on the connection will fail. Data only passes through links, connections, or channels which are in the EST state. and for robustness PSs might enforce a timeout parameter on TMP links and connections, so that they will be deleted if not marked EST after a certain time period.

Link Identifier (*lid*) { *adr,* WORD[2] }

The link identifier is used to uniquely identify a logical link within a connection. It consists of the address of the PS which initially required a link, and a number guaranteed to be unique within that switch.

Logical Link (*link*) { *state, lid, *plink, *p* }

Each connection may have several logical links indicating that data will be received and transmitted from and to the neighboring device indicated by the *plink* reference.‡ The link may be either temporary or established as indicated by the *state* element. There will be only one established link to any given neighbor for a given connection. The *lid* will be used to identify the temporary links, so that return messages get back to the appropriate device. And, if the neighboring device is a CPE, the established logical link will contain an ordered list of channel permissions. Note that this information is not replicated in every PS. Only the PSs neighboring CPEs will have permission lists in their connection links, and these permissions could be different for each link.

Channel (*chan*) { *b, p, state* }

A channel at the PS level is slightly different from that at the BPN level. The additional state information associated with the channel is important for bandwidth allocation. Only channels in the EST state can have data transmitted through them.

Connection (*conn*) { *cid, state, J, *chan, *link* }

This is very similar to the connection structure described for level N. It contains the connection identifier, the state of the connection, and its accessibility. An ordered list of channel parameters is specified, along with a list of logical links which are part of this connection. The dashed boxes in Figure 5.1 represent the connection structures in the sample packet switch.

Figure 5.2 shows an example of a connection structure within a PS. In this case the cid is A3, and the accessibility is N. There are two channels. The first (channel 0) is established

---

‡ Here the * means a reference to an individual data item instead of a list. It is represented in Figure 5.1 by the dashed arrows.

(EST) and requires a 50K bandwidth and has a default permission of TR (all bandwidths in these examples are shown in Megabits/second). The second is temporary (TMP) and requires a 30M bandwidth and has a default permission of R. The connection also has six logical links. The first four are established and so route data to and from four physical links. Of these, two are connected to CPEs. The channel permissions for these two channels are indicated in each CPE (though this information is actually stored within the PS), and the available and total bandwidth for the link is indicated above the physical link itself. The two other established channels are connected to PSs. The available and total bandwidth for these links are indicated. The last two logical links are temporary and are awaiting a reply to come from one of the two physical links they reference. Their unique identifier is shown and indicates they originated in PS number one. Once the reply comes through, they will be deleted, because they reference physical links which other, established logical links also reference.

Packet Switch (*PS*) { *adr, *conn, *plink, rdata* }
Finally, a packet switch contains the relatively static elements of address, a list of physical links, and routing information (*rdata*). It also contains a dynamic list of connection structures.

5.2. Connection Operations

Operations on connections in PSs can be in one of two broad categories: intraswitch operations or interswitch operations. Intraswitch operations are merely primatives for creating, destroying, and modifying the data structures required for representing connections. They affect only the PS on which they are executed. Interswitch operations are messages passed from PS to PS. The execution of intraswitch primatives results from the transmission of these messages. Also, other outgoing, interswitch messages might be produced as a consequence of an incoming message.

Another group of messages and primatives are required: control operations. These are responsible for such functions as loading routing tables and modifying the network topology. Since this is a connection management specification, however, these network management functions are not addressed in detail. Network topology is assumed to be fixed, and routing is abstracted into an unspecified function and also assumed to be static.

5.2.1. Intraswitch Primatives

These primatives are the operations required to manage the connection data contained in a PS. They are always executed upon the request of an incoming message.

Create connection `create_con` (*cid*)
This causes a connection data structure to be allocated. The network may have some default settings for the connection parameters such as accessibility, channel configuration and so forth. In the previous section we had assumed that the defaults were those values appropriate for a simple voice telephone call. The state of the connection upon creation is TMP and no links exist within it.

Modify connection `mod_con` (*cid, data*)
To alter any parameter of a connection, this operation must be performed. Since there are so many different possibilities for modifications, they are not listed here, but rather abstracted into this one operation. Note that these include only the modifications to the connection structure proper, and not the link elements of that structure. Changing the

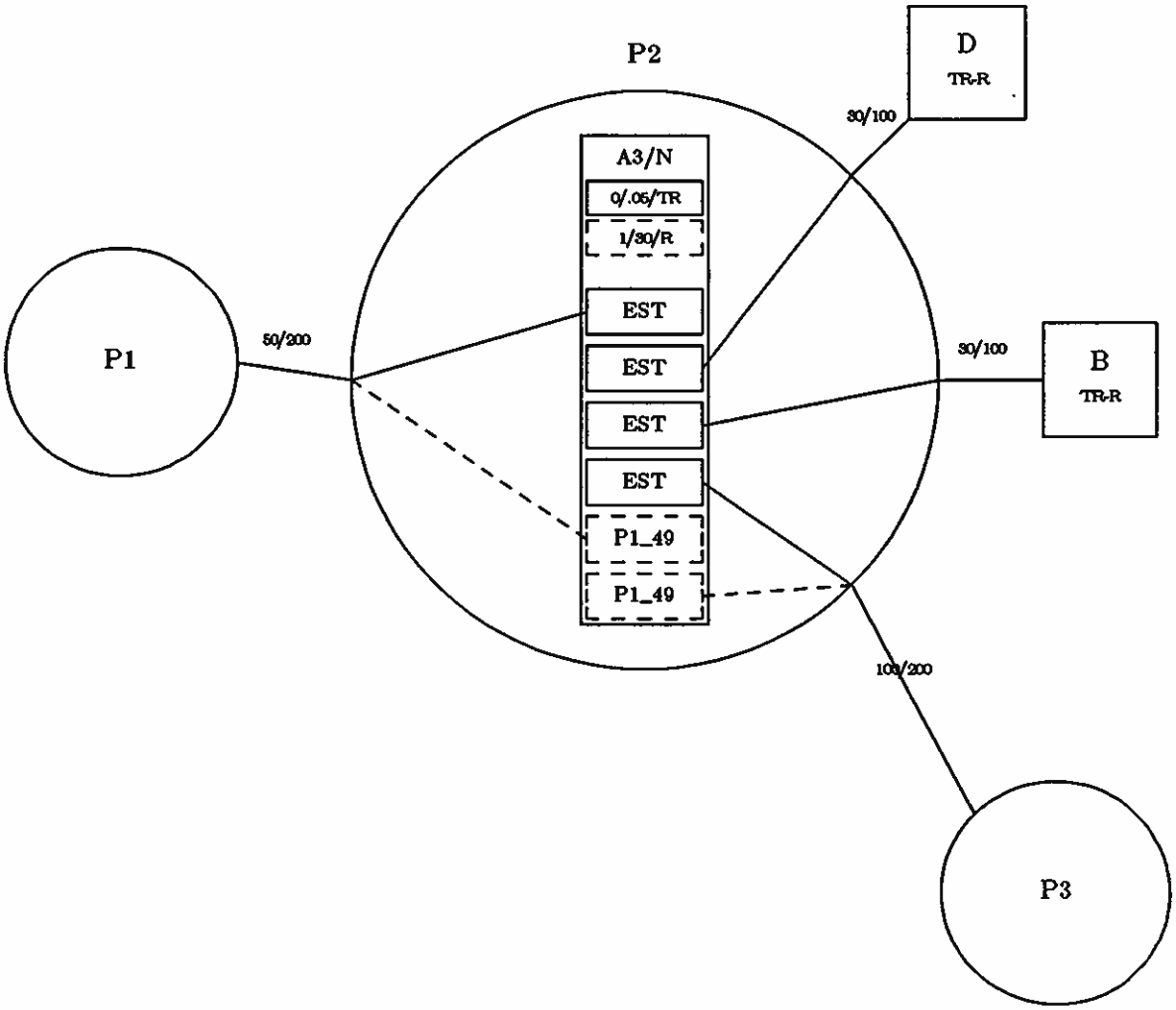


Figure 5.2: Packet Switch Connection Data Structure

state of a connection to EST from TMP would require the use of this function.

Delete connection `del_con` (*cid*)
The connection is deleted from the PS and any memory associated with it is freed. Presumably the links of the connection would have been deleted with the `del_link` operation, but there is no reason why this function could not do that also.

Create link `create_link` (*cid*, **plink*, *lid*)
A logical link is created, within the connection designated by *cid*, and connected to the physical link designated by **plink*. It is given the lid *lid*. The state is TMP, and, if the neighboring device is a CPE, the channel permissions are initialized to the default value specified by the channel structures within the connection.

Modify link `mod_link` (*lid, data*)
Any modifications to a link requires the use of this function. The most frequent modification will be that of changing state and channel permissions.† Bandwidth is allocated and deallocated with other primatives.

Delete link `del_link` (**plink, lid*)
This frees all the memory associated with a link. Data from the connection will no longer flow to the corresponding neighbor.

Allocate bandwidth `alloc_bw` (*b, *plink*)
If the amount of available bandwidth on the given plink is greater than or equal to that requested by this operation, it is reduced (allocated), and the operation succeeds. Otherwise the operation fails.

Free bandwidth `free_bw` (*b, *plink*)
The amount of bandwidth available on the specified plink is increased by the amount indicated.

Route `route` (*adr*)
Routing is abstracted into this function. A destination address is given, and the plink out which the data should travel to reach the destination is returned. A more thorough discussion of the possible routing algorithms can be found in [6].

5.2.2. Interswitch Messages

Interswitch messages are one of three types: single switch, routed, or broadcast messages. These messages pass between CPEs and PSs or two PSs, and result in PS connection data being altered. In the case of routed and broadcast messages similar messages are then transmitted from the affected PS to one or more of its neighbors.

Most of these messages correspond with an operation specified at the BPN level. It should be noted that because of this the names and parameters are similar. One difference is a need for an extra, identifying parameter and possibly other information about the connection within the message. The link id (*lid*) suffices as an identifying parameter. It is used to route the reply message, since the reply might be generated at a distant PS.

5.2.2.1. Single Switch Messages

Single switch messages occur only between CPEs and PSs. If a CPE sends one to a PS, the PS answers directly without sending a further message to another PS. PS to CPE messages are also answered by the CPE directly and do not result in other messages of the same type begin propagated. They are, however, usually a result of an incoming message to the PS, so messages of a different type are usually sent by the PS to other PSs as a reply.

Open a connection `open_con` (*adr, lcn*)
Such a message received from a CPE by the PS results in the creation of a connection structure with a cid formed from the address of the CPE and the lcn on which the connection data will be transferred. A link within the connection to the originating CPE

---

† Altering the corresponding physical line (*plink*) for a given logical link might also be possible. This would, in effect, reroute the connection. While not explored here, this design does not eliminate this technique which

is created, and both the connection and the link are marked EST. The connection is not created, however, if there is insufficient bandwidth or other resources (e.g. PS memory). A response message is sent to the CPE giving the cid and the status of the request (OK or some error indication). By default, the opening CPE is the root CPE, and its neighboring PS is termed the root PS.

Invite to join connection `invite` (*cid, J, m,* $b_1, \ldots, b_m, p_1, \ldots, p_m$)
The PS will send this to a neighboring CPE as the result of an incoming `add_ep` message. The structure should already be created within the PS for this connection, so no other operations are performed. This message contains all the information about the connection. A response is sent by the CPE indicating whether or not it will accept the invitation.

5.2.2.2. Routed Messages

Routed messages are termed such, because they have a specific destination associated with them. This destination may be a PS adjacent to a called CPE, the root CPE of the connection, or the first PS on the path from the source CPE to the root of a connection which contains a copy of the connection structure. Their replies return by the same path the original message used to get to the destination. These messages can be one of two types: connection creating or connection following.

5.2.2.2.1. Connection creating messages

These messages might create connection structures in a given instance, because the origin or destination of the message is not already part of the connection. When they encounter PSs which do not have copies of the connection structure they create a TMP one. These structures are marked EST if the message reaches its destination and a positive reply is received, otherwise they are deleted.

Add an endpoint `add_ep` (*cid, adr*)
This request is initially transmitted from the root CPE to the root PS, where a connection structure must exist from a previous `open_con`. A TMP link is created back to the root and the message is then routed out one of the plinks of the PS towards *adr*. A TMP link is created to this plink, and another `add_ep` is sent out on it. If there is not already an EST link to this plink, the complete configuration of the connection will be included in this new message, otherwise, there is a connection structure at the next PS, so that information can be gotten there.

If there is a connection at this new PS the previous steps are repeated to route the `add_ep` message to another PS. If not, the information within the message is used to create a TMP connection structure, and then a new `add_ep` message is sent as in the previous case.

This continues until the message reaches the last PS (i.e. the one adjacent to the addressed CPE). At this point an `invite` message is sent to the CPE instead of an `add_ep`. If the reply of the CPE is negative, a negative `r_add_ep` message is sent back to the root along the TMP links created on the way out, deleting these links as the message progresses back. If any of the deleted links is the last in a TMP connection structure, that structure is also deleted. The reply finally arrives back at the root, and all

might be useful for congestion control and fault tolerance.

the TMP links the original message created have been deleted.

If the reply from the CPE is positive, a similar procedure is followed. In this case, however, the TMP links are marked EST, and any TMP connection structure is also marked EST. Actually, though, only those TMP links which do not have a corresponding EST link to it are marked EST. The others are deleted. This results in the establishment of only one EST link from a PS to a given destination for a given connection. The "Remote Endpoint Added" and "Endpoint Addition Fails" sections of the scenario following depict the positive and negative cases of this message.

Join a connection `join_con` (*cid, lcn*)

When a CPE sends requests to join a connection, a copy of the connection structure must be found. The neighboring PS (termed the originating PS) creates a TMP connection structure, and then routes a `join_con` message towards the root of the connection. Since the CPE must know the connection's cid to request the join, it knows the root address. TMP links are also created within the TMP connection structures to mark the route taken by the `join_con` message.

Eventually the message will reach a PS where a structure for the connection already exists. This PS checks the accessibility of the connection.‡

If the accessibility is N (nobody can join), a negative `r_join_con` message is returned. The TMP links and connections along the return path are deleted, and the negative reply is sent on to the originating CPE.†

If the accessibility is A (anybody can join) or C (check with root), a positive `r_join_con` reply message is sent back to the originating PS along the path on which the original message came. This message contains the complete configuration information for the connection. The bandwidth required for the connection is allocated as the positive `r_join_con` message returns. If, at any point, insufficient bandwidth exists, the reply message indication is changed to failure. As long as the indication is success, the missing connection information is supplied from the message at each PS, so that the connection structures can be completed. The state of the connection structures, however, is not altered to EST at this time nor are the TMP links deleted.

When the reply reaches the originating PS, an `est_join` message is sent back along the path of TMP links. If the reply message indicated a failure, then the `est_join` message will tear down the TMP links, delete the TMP connections, and deallocate any successfully allocated bandwidth associated with the `join_con` request. If it indicated a success, and the connection accessibility is A, then the opposite occurs. The TMP links and connections are marked EST as the reply to the `est_join` message returns to the originating CPE.

---

‡ The connection structure found might also be a TMP structure, and so not have the accessibility information. If this is the case, a `join_con` message is propagated as if the structure were not found, though the TMP links are created within the existing TMP structure.

† It might be the case here, and with many of the operations explained in this section, that another request for this connection had passed through one of the PSs involved in this temporary path. It also might have succeeded and thus caused the connection to be marked EST at this PS. Note that this other request would have created TMP links of its own which would not conflict with those of the first call. In this case, when the first call passes back through, deleting its links, it would not delete the EST connection structure. Connection structures (both EST and TMP) are only deleted when the final link within them is deleted.

If, however, the accessibility is C (check with the root), when the `est_join` message reaches the PS where the connection was found, this PS generates a `join_req` message, and sends it towards the root of the call. Upon the reply to the `join_req` message, a positive or negative `r_est_join` message is sent as before, depending upon the answer, establishing or deleting the connection as it returns.

Note that this algorithm is guaranteed to succeed. At worst, the first EST connection structure will be found at the PS adjacent to the root CPE, because the `join_con` message is routed towards the root. At best it will find the connection already established in the PS adjacent to the requesting CPE. In practice it will find it at some PS between these points. Even if the connection does not exist, this will be indicated when the message gets to the PS adjacent to the purported root CPE, and a negative response will be returned.

An example of the use of this, the `est_join`, and `join_req` messages (whose descriptions follow) can be found in the "Additional Endpoint Requests to be Added" part of the scenario in this section.

Get connection information `get_info` (*cid*)
This is similar to the `join_con` message in that it is sent towards the root of the connection and progresses until it finds an EST connection structure. The accessibility is also checked at this point, and a failure is returned if it is N, or a success is returned if it is A. A special `join_req` message is sent to the root if it is C, and a success or failure is returned based upon the reply from the root indicating access or not. If the reply is to be successful, the connection information is returned in the `r_get_info` message, so there is no need for the two stages required with the `join_con` message.

5.2.2.2.2. Connection following messages

These messages follow existing connection structures towards their destination, because their origin and destination is already part of the connection. TMP links are still created, however, so that replies can return, and these replies delete the TMP links created by the original message.

Establish a join to a connection `est_join` (*cid, endp*)
This message is different from most of the others in that it is not generated or destined for a CPE. It is purely an internal network message to be sent between PSs. It is used to establish a connection path initiated by a CPE outside the call by a `join_con` message. It follows TMP links created by the original `join_con` message, and it causes the `join_req` message to be sent if the connection accessibility is C. Its reply, and not that of the `join_con` is what changes the state of the connection and links to EST. This intermediate message is required, because the originating CPE does not know the bandwidth required by the call before contacting it.‡

Request for join to connection `join_req` (*endp*)
This is generated as a result of a `est_join` message finding a connection structure with

---

‡ This is in contrast to the `add_ep` message which originates from the root. Since the bandwidth is known, it can cause its allocation while on the way to its destination, returning failure and deleting the TMP connection structures immediately upon encountering a point where insufficient resources exist. In the present case, however, the detection of insufficient resources occurs on the reply's return, so another message is necessary to delete the TMP connection structures (or mark them EST in the case of success).

an accessibility of C. It contains information about the requesting CPE, and is routed along an established connection to the root. The root then uses this information to grant or deny access to the connection, and then returns a reply message.

Delete an endpoint `del_ep` (*adr, lcn*)
This message will be routed towards the CPE at *adr*. Once at the PS adjacent to this CPE, the link of the connection which corresponds to the *lcn* of that CPE is deleted. An `r_del_ep` message is then generated for return to the root. If there is only one remaining EST link and it is to another PS, that link and the connection structure are deleted. The reply message contains information concerning this deletion, so the EST link in the next PS which corresponds to the deleted link can, itself, be deleted.* An example of this message can be found in the "Endpoint is Deleted" section of the scenario.

Modify channel permissions `mod_per` (*cid, adr, lcn,* $p_1, \ldots, p_n$)
This root generated message causes the modification of the channel permissions for a given endpoint in the connection. The old permissions in the link from the PS neighboring the specified CPE are replaced by the new ones in the message. Again this message, like the others in this section, is routed along established links towards the designated CPE, and the reply follows the TMP links, created on the outgoing journey, back to the root. This message is depicted in the "Video Lecture is Established" section of the scenario.

Disconnect from a connection `dropout` (*cid*)
This message is generated by a CPE which would like to disconnect from a connection. It is similar to the `del_ep` message. Since the originating CPE is already part of the connection, its neighboring PS will have an EST connection structure. The PS must count the number of links within that connection. If there are more than two (i.e. another CPE or PS is relying on this PS's connection for its data), then the link to the originating CPE is merely deleted and a reply is sent to the CPE. If, however, there are only two links then the `dropout` message is propagated away from the originating CPE, causing the same operations to be performed until it reaches a PS with more than two links in the connection structure or the PS adjacent to the root CPE. It deletes the EST links along the way, but creates TMP links, and when it reaches the final PS, the `r_dropout` message produced follows these TMP links back to the originating CPE, deleting the connection structures in the intervening PSs. The "Endpoint Drops Connection" section of the scenario contains an example of this message.

5.2.2.3. Broadcast Messages

Broadcast messages are not broadcast to the entire network, but rather are sent to every PS involved in the given connection. These messages will modify the data that is replicated in all the connection structures of a given connection.

---

* Two things should be noted here. First, the deletion takes place after the reply message is sent, so that it can be properly sent (it uses the link). Second, it is assumed that routing algorithms cause the connection structures and links of a connection to form a tree within the network of PSs, and that connection following, routed, interswitch messages always follow these existing, established links. If this second statement were not true, then a `del_ep` message could come in from a different PS than that of the incoming data from the root, and, more importantly, the reply would be directed to a different PS than that on the other end of the deleted link. This case could be handled, however, by a new message which would only affect the one PS at the other end of the deleted link. It would be sent to that PS, while the `r_del_ep` message proceeded back along the path on which it came.

The algorithm used to broadcast the message is straightforward, because the connection forms a tree within the network (this tree is termed the connection tree, PSs with a link to CPEs and only one other PS are the leaves, and the root is the root PS). When a message comes into a PS with only one outgoing link, it is merely propagated along that one link (creating a temporary link for the reply, of course). The reply then comes back through that PS in a manner like that of a routed message. If there is more than one link, a new message is generated with a different identification for each outgoing link. The reply for that PS is then sent only when a reply is received for all the new messages generated.

Open a channel `open_chan` (*cid, n, b, p*)

Opening a new channel in a connection requires allocating bandwidth throughout the network. The available bandwidth is checked when the `open_chan` comes into the PS to see if there is enough for the new channel. If not, a reply is sent immediately indicating a failure, and no message is propagated further. If there is, and the message can be propagated further (i.e. this PS is not a leaf in the connection tree) then the bandwidth is allocated, and a message is sent to the following PSs. The reply is returned only when the replies to the propagated messages are received. The reply is successful if all of the returned replies are, and a failure otherwise. If there is sufficient bandwidth, but this is a leaf PS then a successful reply is returned immediately.

Before any successful reply is returned, however, the connection structure is updated to include the new channel's information. The state of the new channel is TMP.

Eventually, all the PSs involved in a connection will encounter an `open_chan` message and will have replied, and the root PS will get replies from its adjacent PSs. If all replies are successful then there is enough bandwidth throughout the network to handle the new channel, otherwise there is not enough, so the request of the root CPE (for the new channel) cannot be granted.

If the reply to the CPE would be fail, then the PS sends out a `close_chan` message, so that all those PSs that have established the TMP channels can delete them. If the reply is to be successful, then the PS will send out the `est_chan` to all the PSs so that they can change the state of the TMP channel to EST. In this case the reply to the original `open_chan` is sent to the CPE only after all the replies to the `est_chan` message are received.

Examples of this and the following `est_chan` message can be found in the "Video Lecture is Established" section of the scenario.

Establish a channel `est_chan` (*cid, n1, n2*)

This internal message is used to change TMP channels to EST for purposes of creating or modifying the bandwidth required for a channel. The message may cause one of three different results depending upon the values of the channel indication parameters (*n1* and *n2*). All changes occur as the replies to this message return.

If the two parameters are the same, then the state of the indicated channel inside the connection structures of the call will be changed from TMP to EST. This type of `est_chan` message is used in conjunction with the `open_chan` message.

If the bandwidth required by channel *n1* is less than that by channel *n2* then channel *n1* is deallocated, and channel *n2* is renamed *n1* and put into the EST state. This is used after a successful `mod_chan` resulting in the raising of the channel *n1*'s bandwidth

Otherwise the channel indication parameters differ and the bandwidth required by channel *n1* is greater, so channel *n1* is deleted after the difference between the bandwidths of the two channels is added to that available on all the physical links corresponding to the connection. This is used after a failed `mod_chan` which requested an increase of bandwidth for channel *n2*, and results in the deallocation of that increase.

Close a channel `close_chan` (*cid, n*)

The `close_chan` message causes the resources associated with a channel to be deallocated. The channel within the connection structure, and the bandwidth associated with it are freed when the reply message returns from the leaves of the connection tree. The memory required to store the information in the links to CPEs is also freed.

Modify a channel `mod_ch` (*cid, n, data*)

This message works similarly to the `close_chan` message if the bandwidth of the channel is being lowered or remaining the same. Instead of freeing the channels, the information associated with them is modified. If the bandwidth is being lowered then the difference is returned to the plinks in each PS involved in the connection.. If, however, the bandwidth is being raised, this message is more like the `open_chan` message in that two stages are necessary. The additional bandwidth is allocated as a result of this message, and the corresponding channel structures are put in the TMP state.‡ Also, an additional channel is allocated (with a new channel number *newn*). Its bandwidth requirements are set to the new requested level, the other parameters are the same, and it is in the TMP state.

If, upon return to the root PS, a success is indicated, then an `est_chan` (*cid, n, newn*) is sent out. This causes the new channel structures, with the higher bandwidth, to replace the old.†

Otherwise, a failure is indicated, so an `est_chan` (*cid, newn, n*) is sent. This causes the deletion of the new channel structure and the deallocation of the added bandwidth. This occurred because there was at least one PS within the network which could not honor the request for more bandwidth.

In all cases the status of the operation is returned to the root CPE. An example of this and the following `mod_acc` message is included in the "Video Lecture is Established" section of the scenario.

Modify connection accessibility `mod_acc` (*cid, J*)

This message behaves like a `mod_chan` message where the bandwidth is not being raised. It differs in that it modifies the connection accessibility indicator in the connection structures. Replies are returned like the other messages of this type, and failures should be rare.

Close a connection `close_con` (*cid*)

This message causes all the resources allocated to a connection to be freed, and so deletes the connection from the network. This deletion occurs as the reply is returning to the root

---

‡ The implication here is that no data will be sent on this channel during this stage. This may not be acceptable in which case, a new state for the channel would be required.

† This replacement might be as simple as updating the bandwidth in the original structure and deleting the new one, or as complicated as deleting the original one and renaming the new one. This last technique was suggested in the section describing the `est_chan` message.

CPE. An example of this message is contained in the "Connection is Closed" section of the scenario.

5.3. Scenario

The various steps involved in establishing and terminating a video lecture demonstrate most of the features of the operations specified at the PS level. To show some additional features we shall construct a scenario where a simple telephone call is turned into a conference call between three parties, and then finally into a video lecture.*

The figures associated with this scenario usually come in pairs. The first lists the operations performed and messages sent to place the network in the configuration represented by the diagram in the second. In some cases the diagram is not given because it would duplicate that in another figure. Note that in this example we are only describing one connection. A typical PS would have many connections in progress and so many connection structures allocated at a given time.

The format of the operations and messages listing is simple. Messages are shown with a label of the form SRC→DST: where SRC represents the source of the message and DST the destination. Operations have a label of the form PS: where PS designates the PS in which the operations occur.

5.3.1. Connection is Opened

The initiating CPE sends an `open_con` connection to its adjacent PS. This results in the connection structure being allocated and established. The default connection parameters (accessible by no one, and one channel of 50K bandwidth and TR permission) are useful for a voice-grade telephone call. Everything is marked established and the bandwidth on the CPE to PS line is allocated. Note that in this example only full megabit units of allocated bandwidth are indicated. The open is shown in Figure 5.3a and Figure 5.3b.

5.3.2. Remote Endpoint is Added

The next step in the scenario is the addition of another endpoint to the connection so actual communication can occur. This uses the two stage procedure outlined in the level N section. The root issues an `add_ep` request to the network and the network causes an `invite` to be sent to the destination CPE. This CPE then replies to the `invite`, causing the network to reply to the `add_ep`.

Figure 5.4a and Figure 5.4b show the contacting of a remote CPE. When the `add_ep` message gets to P1, a temporary link back to the root CPE is created within the established connection structure. The physical link used to reach the destination (B) is determined, and bandwidth is allocated on that link, because this is the first time that link has been used in this connection. Another TMP link is created to PS2. The TMP links have the lid P1_13. Further interswitch messages will have this identifying label.

---

* Think of this as two instructors conversing and then deciding that they require the assistance of a third to finalize an idea. The three of them then decide that the conclusions of their discussion are so important that they must immediately share them with their students.

A→P1:	`open_con(A, 3)`

P1:	`create_con(A3)`
	`create_link(A3,P1_01)`
	`alloc_bw(.05,A)→OK`
	`mod_con(A3,EST)`
	`mod_link(P1_01,EST)`

P1→A:	`r_open_con(A3,OK)`

Figure 5.3a: Connection is Opened (Operations)

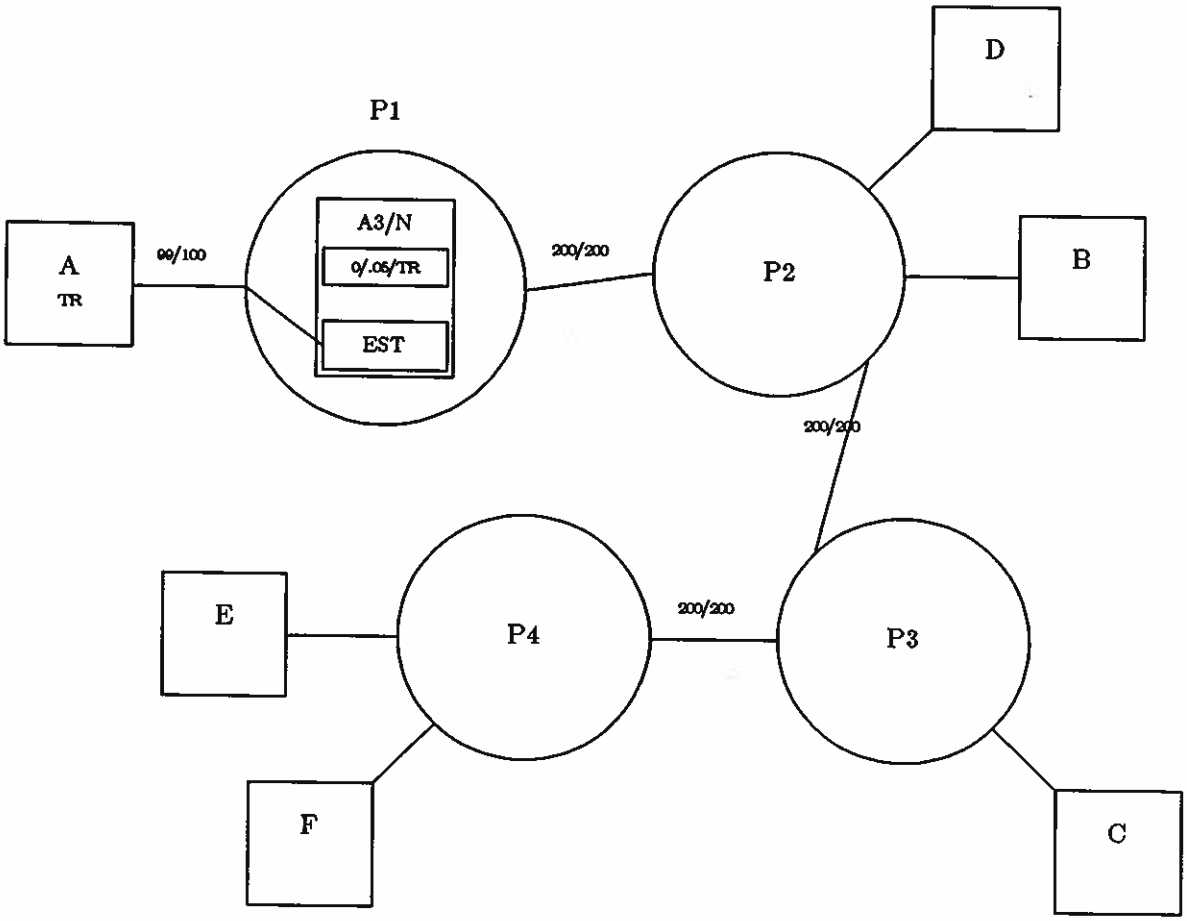


Figure 5.3b: Connection is Opened (Diagram)

At PS2 there is no copy of the connection structure, so the information which was included in the `add_ep` message from PS1 is used to create one. It is left in the TMP state, because the contact has not yet succeeded. A TMP link back to PS1 is also created, so the reply message may return to the originator. Bandwidth is allocated on the physical link to B and a TMP logical link is created. Finally an `invite` message is sent from PS2 to B.

B then responds positively to the invitation, in this case. The connection structure in PS2 is marked EST, as are the two logical links within that structure. A reply to the `add_ep` message is then sent to PS1 causing the TMP link to PS2 to be marked EST A successful reply is then sent to the CPE A. These messages and their result are shown in Figure 5.5a and Figure 5.5b.

A→P1:	`add_ep(A3,B)`

P1:	`create_link(A3,A,P1_13)`
	`route(B)→P2`
	`alloc_bw(.05,P2)→OK`
	`create_link(A3,P2,P1_13)`

P1→P2:	`add_ep(P1_13,A3,B,`*A3-info*`)`

P2:	`create_con(A3,`*A3-info*`)`
	`create_link(A3,P1,P1_13)`
	`alloc_bw(.05,P1)→OK`
	`alloc_bw(.05,B)→OK`
	`create_link(A3,B,P1_13)`

P2→B:	`invite(P1_13,A3,`*A3-info*`)`

Figure 5.4a: Remote Endpoint is Added (Operations)

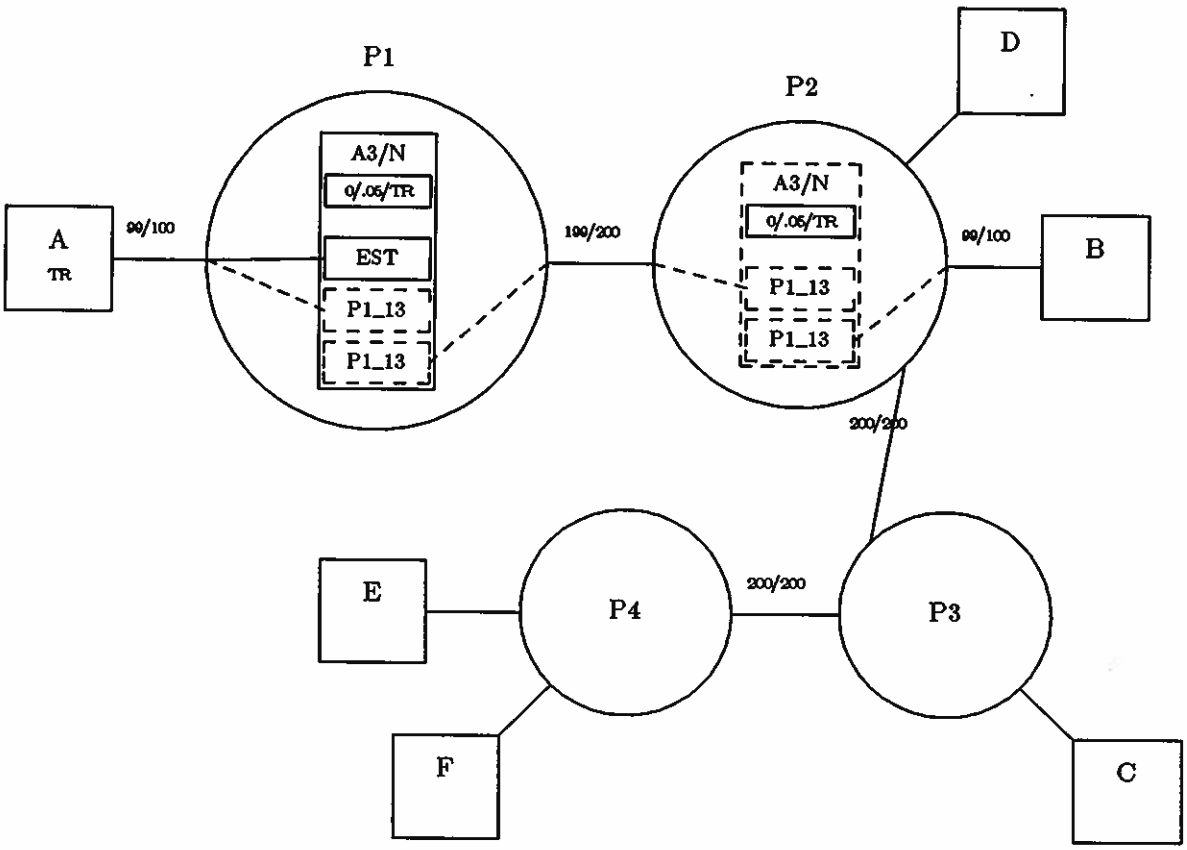


Figure 5.4b: Remote Endpoint is Added (Diagram)

B→P2:	`r_invite(P1_13,A3,OK)`

P2:	`mod_con(A3,EST)`
	`mod_link(A3,B,P1_13,EST)`
	`mod_link(A3,P1,P1_13,EST)`

P2→P1:	`r_add_ep(P1_13,A3,OK)`
P1:	`mod_link(A3,P2,P1_13,EST)` `del_link(A3,A,P1_13)`
P1→A:	`r_add_ep(A3,OK)`

Figure 5.5a: Connection Completion (Operations)

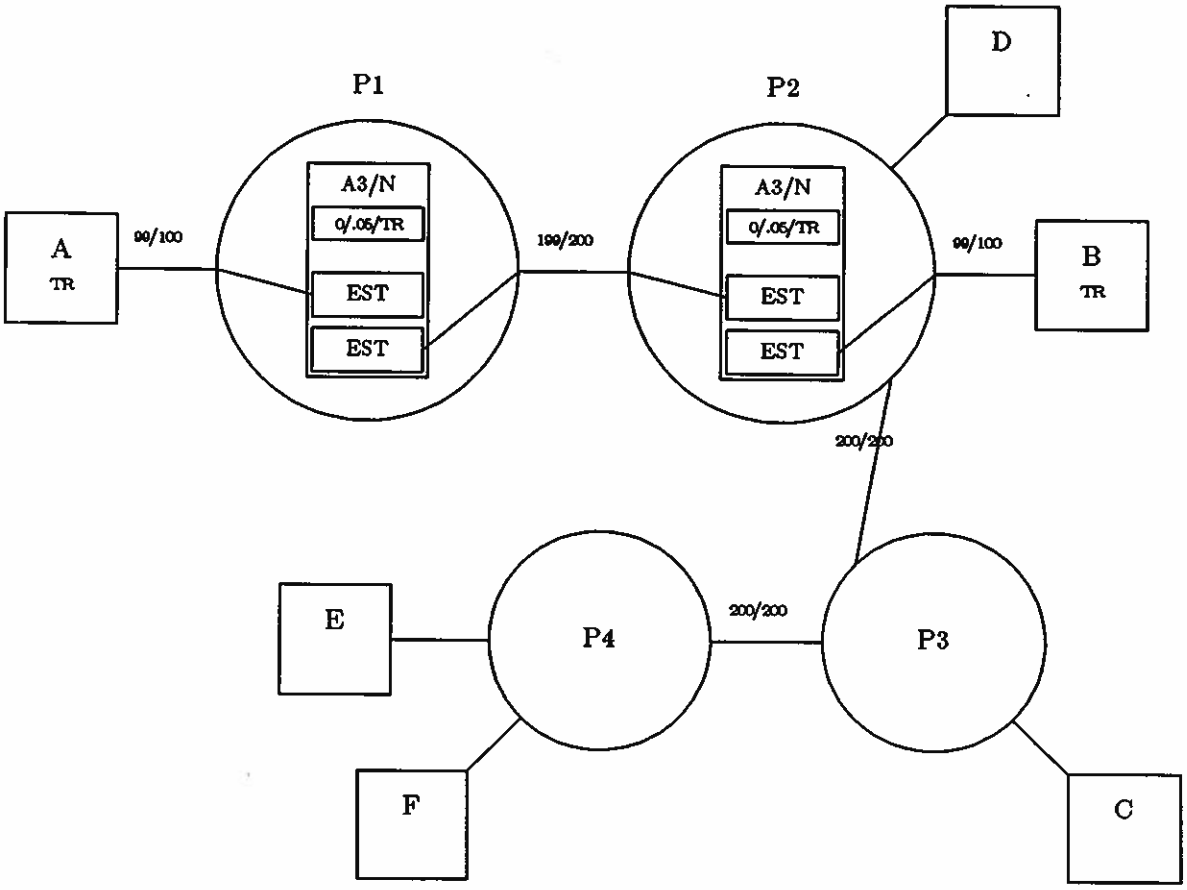


Figure 5.5b: Connection Completion (Diagram)

5.3.3. Third Party is Added to the Connection

Figure 5.6a and Figure 5.6b show an attempt to add a third party to the connection. In this case the TMP links are created just as before in P1, though they will most likely have a different lid. Note that the EST links are not affected nor is extra bandwidth allocated from P1 to P2, because an EST link already exists. Once at P2 the message causes two more TMP links to be created within the existing connection structure, and bandwidth to be allocated on the physical link to D. An `invite` message is then sent to D.

Figure 5.7a and Figure 5.7b show the completion of the third party addition. This is similar to the completion in the previous section. D replies favorably to the invitation causing the link to it to be marked EST. The link from P2 to P1 is not required, because an EST link already exists, so it is deleted instead. The same occurs to the two TMP links in P1, and the reply to the `add_ep` message is sent to the root. Now A, B, and D are communicating over a voice conference call.

A→P1:	`add_ep(A3,D)`

P1:	`create_link(A3,A,P1_92)`
	`route(D)→P2`
	`create_link(A3,P2,P1_92)`

P1→P2:	`add_ep(P1_92,A3,D)`

P2:	`create_link(A3,P1,P1_92)`
	`alloc_bw(.05,D)→OK`
	`create_link(A3,D,P1_92)`

P2→D:	`invite(P1_92,A3,`*A3-info*`)`

Figure 5.6a: Add a Third Party (Operations)

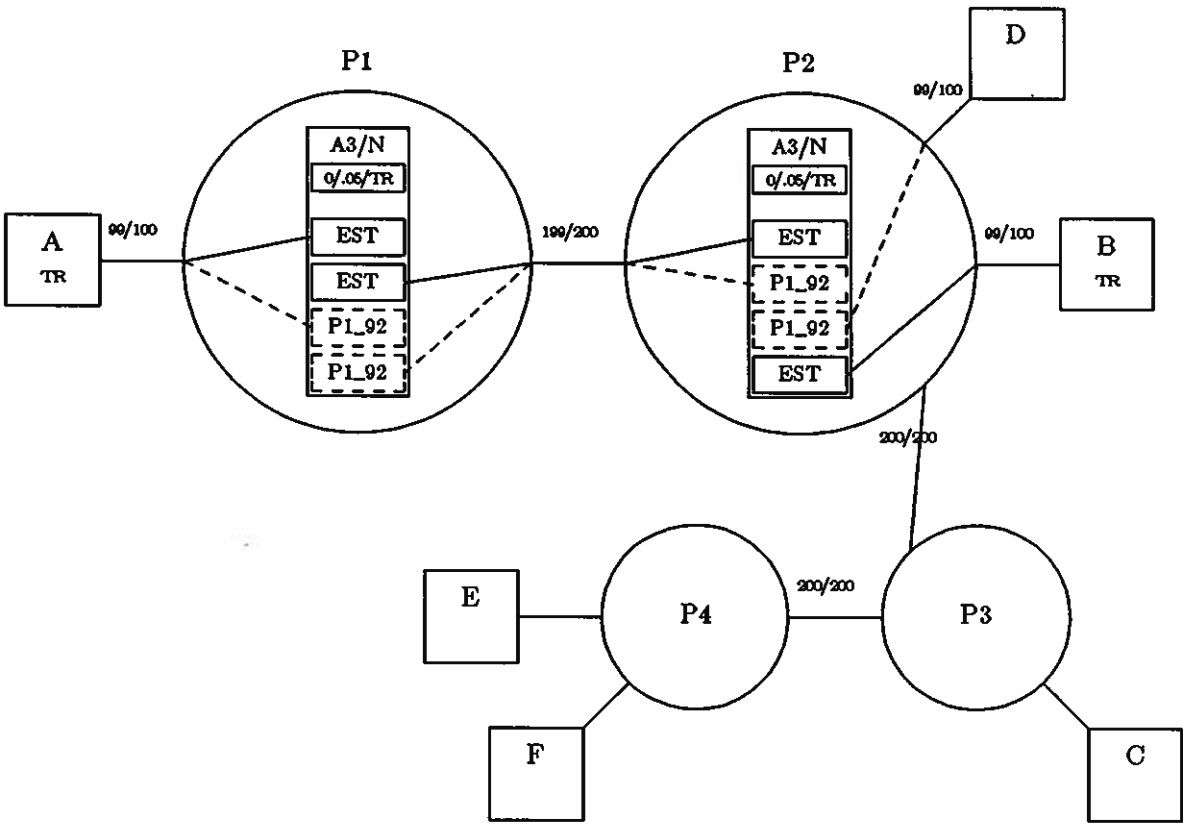


Figure 5.6b: Add a Third Party (Diagram)

D→P2:	`r_invite(P1_92,A3,OK)`
P2:	`mod_link(A3,D,P1_92,EST)` `del_link(A3,P1,P1_92)`
P2→P1:	`r_add_ep(P1_92,A3,OK)`
P1:	`del_link(A3,P2,P1_92)` `del_link(A3,A,P1_92)`
P1→A:	`r_add_ep(A3,OK)`

Figure 5.7a: Complete Third Party Addition (Operations)

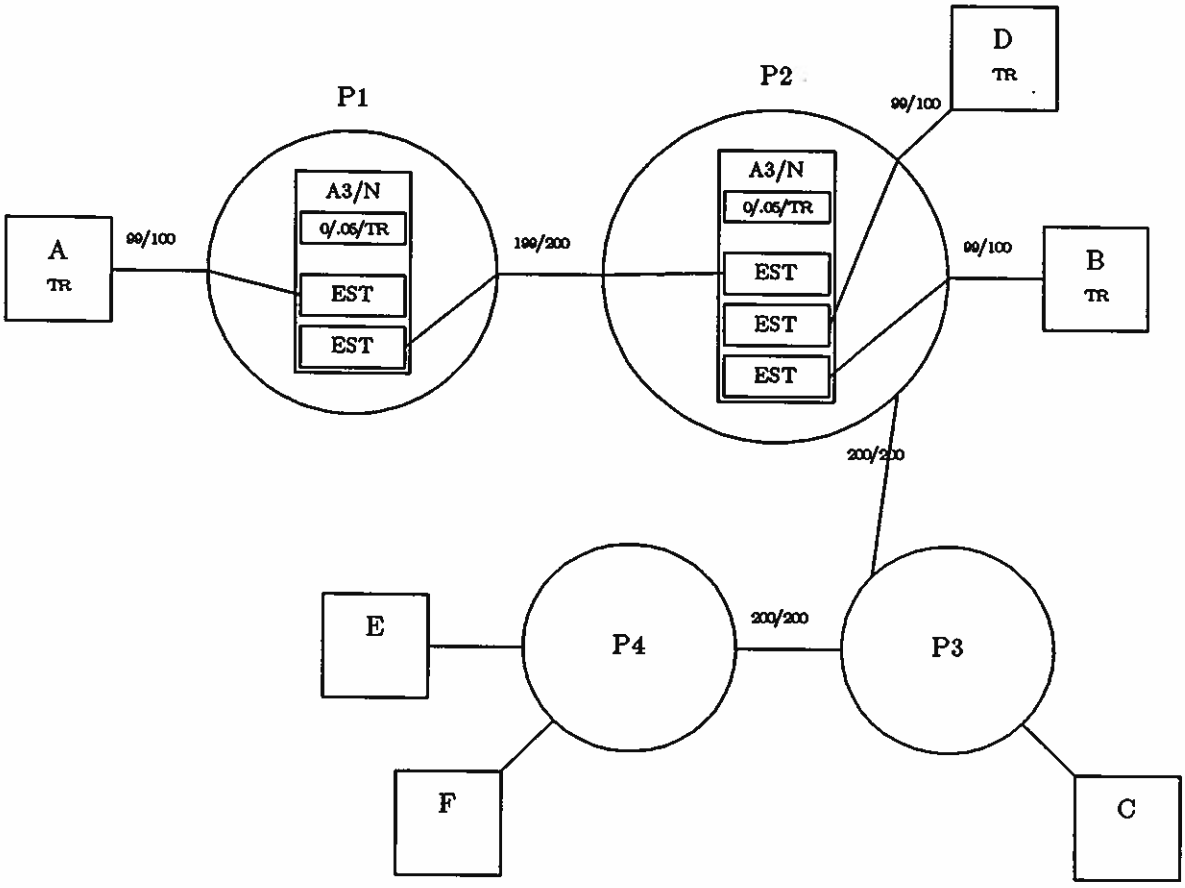


Figure 5.7b: Complete Third Party Addition (Diagram)

5.3.4. Video Lecture is Established

Establishing a video lecture from a simple telephone call requires three steps. The video lecture channel must be opened. The telephone channel will be used for the upstream audio questioning channel. The permissions for the current participants in the call must be altered. And the accessibility of the call must be changed, so that others, with the roots permission, can join the lecture.

Two stages are required to open the video channel. The first stage, shown in Figure 5.8a and Figure 5.8b, involves the allocation of the bandwidth throughout the connection. A TMP channel structure is created in all the PSs involved in the connection. The extra bandwidth is allocated on all the connection links, and a reply finally gets back to the root PS indicating if all the links can supply the required bandwidth. Data still does not pass over this channel, but the voice call can proceed while this occurs.

A→P1:	`open_chan(A3,1,30,R)`

P1:	`create_link(A3,A,P1_57)`
	`alloc_bw(30,A)→OK`
	`alloc_bw(30,P2)→OK`
	`mod_chan(A3,`*`newchan`*`,1,30,R)`
	`create_link(A3,P2,P1_57)`

P1→P2:	`open_chan(P1_57,A3,1,30,R)`

P2:	`create_link(A3,P1,P1_57)`
	`alloc_bw(30,P1)→OK`
	`alloc_bw(30,B)→OK`
	`alloc_bw(30,D)→OK`
	`mod_chan(A3,`*`newchan`*`,1,30,R)`
	`del_link(A3,P1,P1_57)`

P2→P1:	`r_open_chan(P1_57,A3,1,OK)`
P1:	`del_link(A3,P2,P1_57)`

Figure 5.8a: Open Video Channel (Operations)

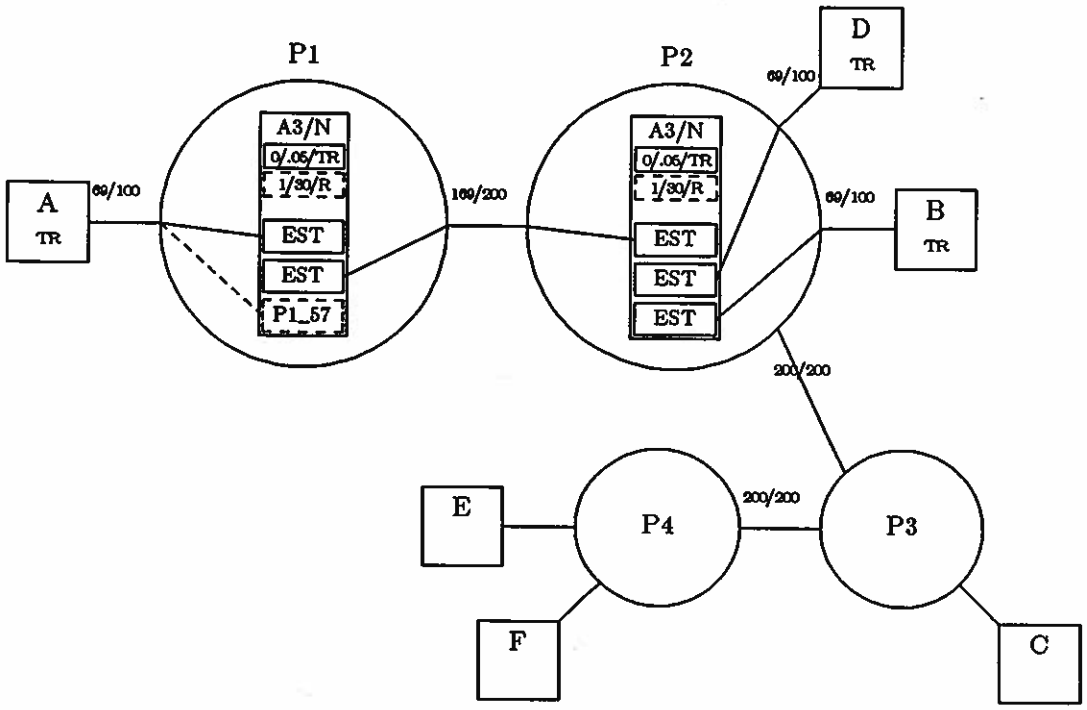


Figure 5.8b: Open Video Channel (Diagram)

The second stage is the establishment of the channel. Once the root gets positive responses from all its neighbors, it sends out an `est_chan` message to mark all the channels EST. This occurs as the reply to this message flows back towards the root. At this point too, the channel permissions for all the CPEs are modified to reflect the default permissions for the new channel. The permissions for other channels are not altered. Also all TMP links created for this message are deleted. Figure 5.9a and Figure 5.9b show the results of this and so depicts an established video channel.

Notice, though, that the CPEs each have only receive (R) permission for this channel, so there is no data being sent. Figure 5.10a shows the operations and messages required, so that one of the participants can transmit on this channel and so be the lecturer. In this case it happens to be the root (A).

It would be nice if anybody who wanted could join the lecture, though it should be limited to those who have permission. Figure 5.10b shows the operations and messages required to set the accessibility of the connection to C (check with root). Figure 5.10c then shows the diagram of the result of channel permission modification and accessibility modification. Thus it depicts an established video lecture.

P1:	`create_link(A3,P2,P1_57)`
P1→P2:	`est_chan(P1_57,A3,1,1)`

P2:	`create_link(A3,P1,P1_57)`
	`mod_con(A3,`*`chanEST`*`,1)`
	`mod_link(A3,B,`*`chan_per`*`,1,R)`
	`mod_link(A3,D,`*`chan_per`*`,1,R)`
	`del_link(A3,P1,P1_57)`

P2→P1:	`r_est_chan(P1_57,A3,1,OK)`

P1:	`del_link(A3,P2,P1_57)`
	`mod_con(A3,`*`chanEST`*`,1)`
	`mod_link(A3,A,`*`chan_per`*`,1,R)`
	`del_link(A3,A,P1_57)`

P1→A:	`r_open_chan(A3,1,OK)`

Figure 5.9a: Establish Video Channel (Operations)

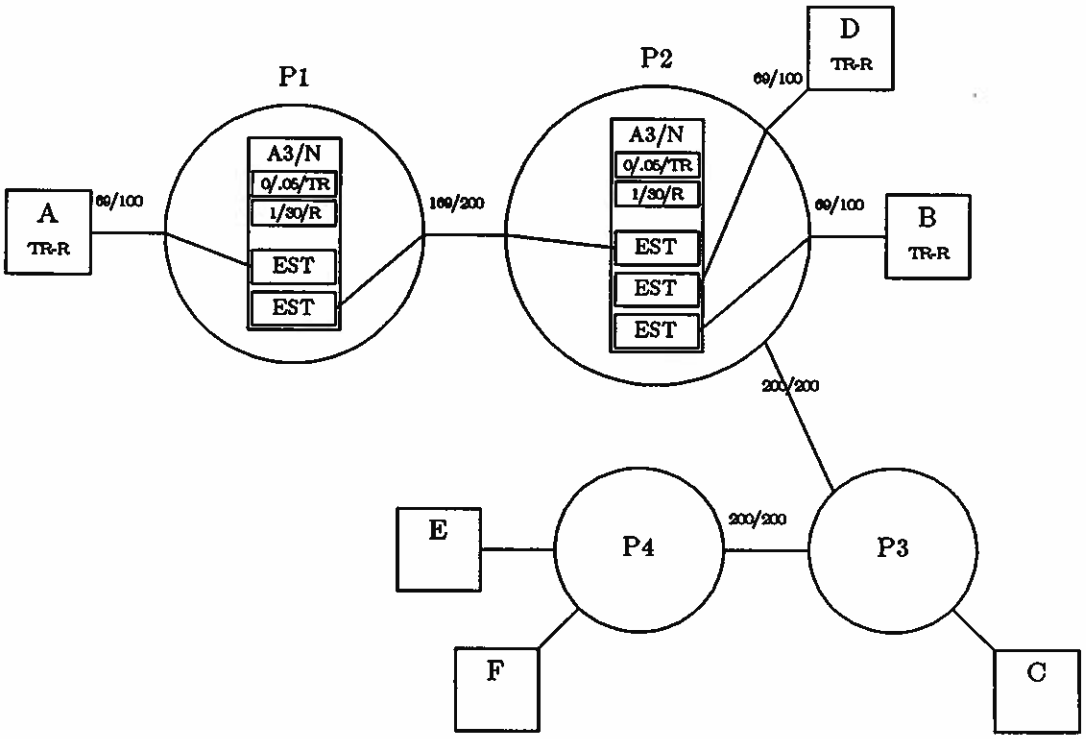


Figure 5.9b: Establish Video Channel (Diagram)

A→P1: `mod_per(A3,A,3,R,T)`

P1:
```
create_link(A3,A,P1_98)
mod_link(A3,B,chan_per,O,R,1,T)
del_link(A3,P2,P1_98)
```

P1→A: `r_mod_per(A3,A,3,OK)`

Figure 5.10a: Modify Video Channel Link Permission

A→P1:	`mod_acc(A3,C)`
P1:	`create_link(A3,A,P1_77)` `create_link(A3,P2,P1_77)`
P1→P2:	`mod_acc(P1_77,A3,C)`

P2:	`create_link(A3,P1,P1_77)`
	`mod_con(A3,`*Accessibility*`,C)`
	`del_link(A3,P1,P1_77)`

P2→P1:	`r_mod_acc(P1_77,A3,OK)`

P1:	`del_link(A3,P2,P1_77)`
	`mod_con(A3,`*Accessibility*`,C)`
	`del_link(A3,A,P1_77)`

P1→A:	`r_mod_acc(A3,OK)`

Figure 5.10b: Modify Connection Accessibility

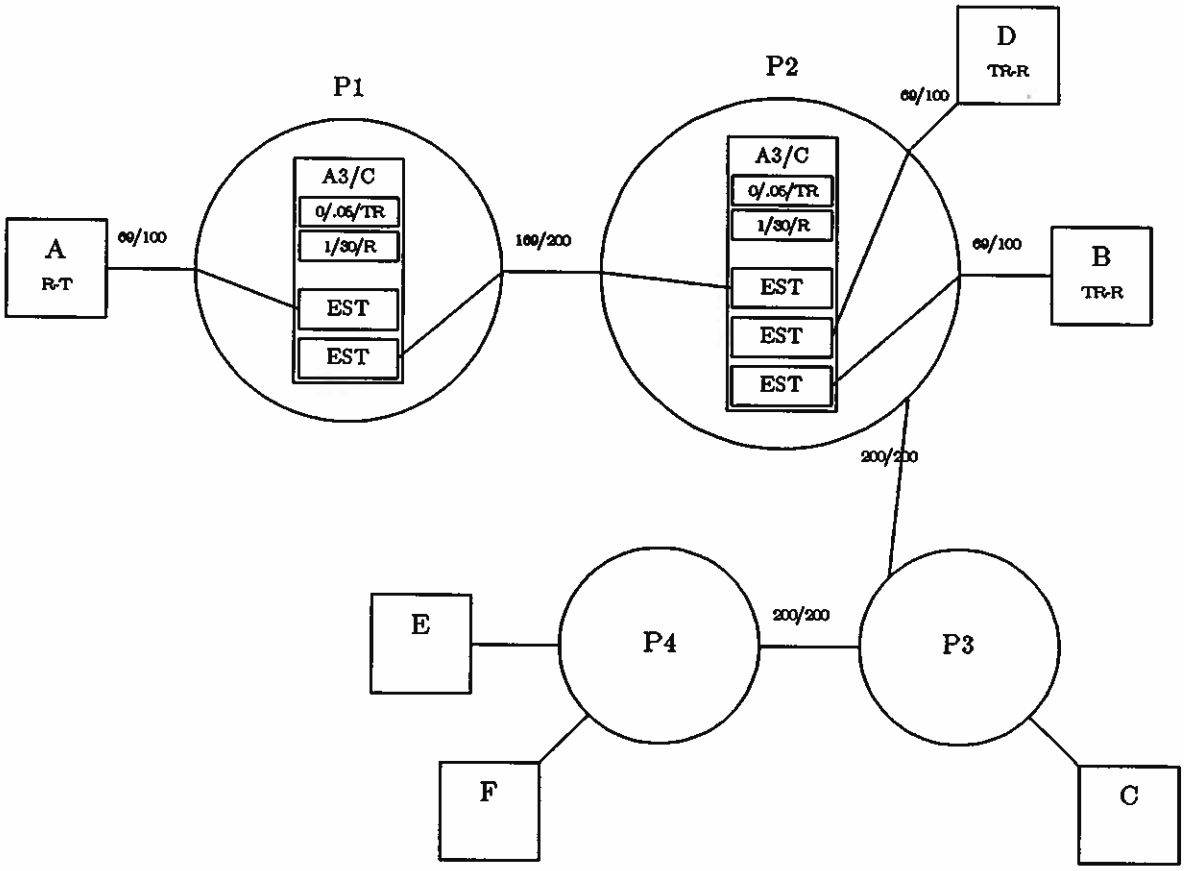


Figure 5.10c: Modify Video Channel Link Permission and Accessibility

5.3.5. Endpoint Addition Fails

The next step is to add a participant to the lecture. In this case C is chosen, but, when contacted, indicates that it does not wish to join. Figure 5.11a shows the operations needed to contact the remote endpoint. The TMP links and connection structures are set up and the `invite` is sent just as with previous `add_ep` examples. Figure 5.11b depicts the state just after the invitation is sent. Since the reply is negative, however, the connection is not established, so all the TMP connection structures and links are merely deleted. Figure 5.11c shows the failed reply message operations. The state of the connection after this is the same as is show in Figure 5.10a.

5.3.6. Additional Endpoint Requests to be Added

To join an in-progress connection requires finding an instance of the connection, allocating the bandwidth and other resources, and requesting permission from the root (if the accessibility is C). In the current example, we have F attempt to join the connection. The PS neighboring F

A→P1:	`add_ep(A3,C)`

P1:	`create_link(A3,A,P1_46)`
	`route(C)→P2`
	`create_link(A3,P2,P1_46)`

P1→P2:	`add_ep(P1_46,A3,C)`

P2:	`create_link(A3,P1,P1_46)`
	`route(C)→P3`
	`alloc_bw(31,P3)→OK`
	`create_link(A3,P3,P1_46)`

P2→P3:	`add_ep(P1_46,A3,C)`

P2:	`create_con(A3,`*A3-info*`)`
	`create_link(A3,P2,P1_46)`
	`alloc_bw(31,P2)→OK`
	`alloc_bw(31,C)→OK`
	`create_link(A3,C,P1_46)`

P3→C:	`invite(P1_46,A3,`*A3-info*`)`

Figure 5.11a: Attempt to Add Fourth Party Operations

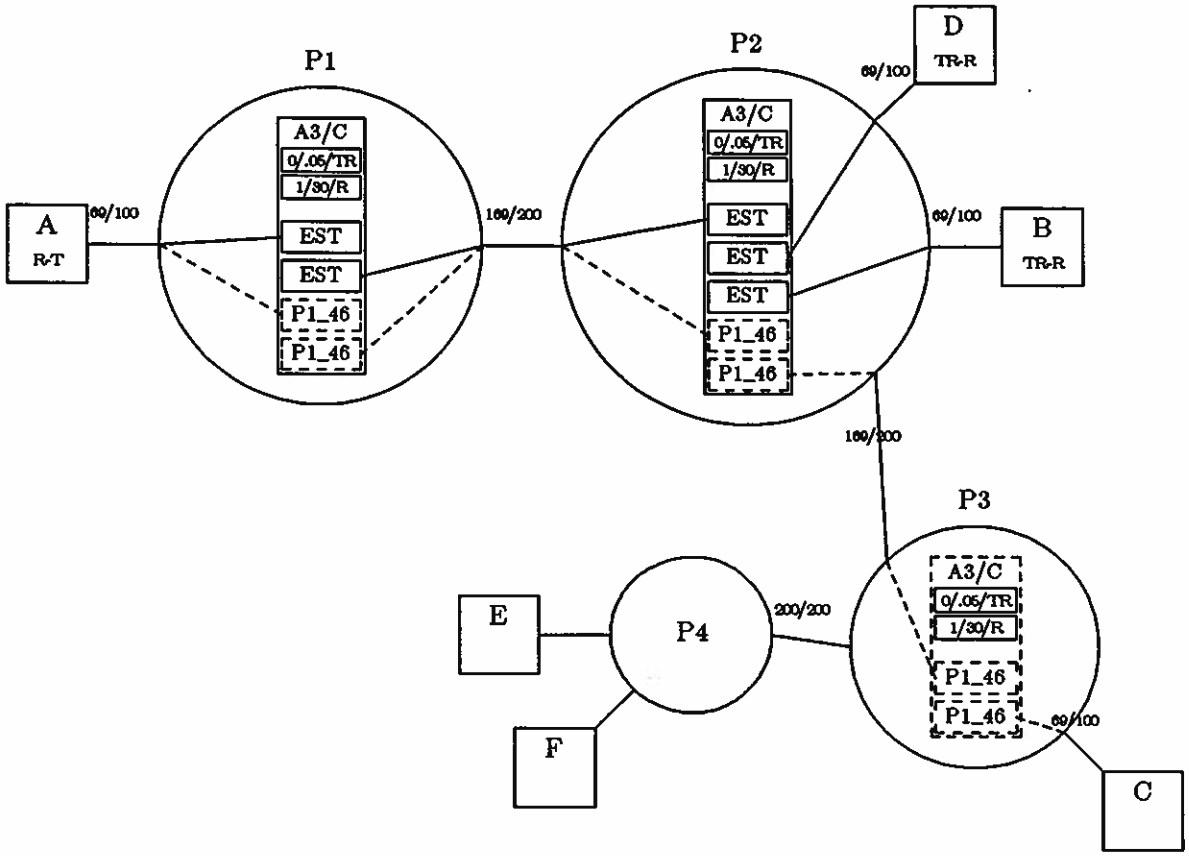


Figure 5.11b: Attempt to Add Fourth Party Diagram

```
C→P3:       r_invite(P1_46,A3,FAIL)

P3:         del_link(A3,C,P1_46)
            free_bw(31,C)
            free_bw(31,P2)
            del_link(A3,P2,P1_46)
            del_con(A3)

P3→P2:      r_add_ep(P1_46,A3,FAIL)

P2:         del_link(A3,P3,P1_46)
            free_bw(31,P3)
            del_link(A3,P1,P1_46)

P2→P1:      r_add_ep(P1_46,A3,FAIL)

P1:         del_link(A3,P2,P1_46)
            del_link(A3,A,P1_46)

P1→A:       r_add_ep(A3,FAIL)
```

Figure 5.11c: Fourth Party Addition Failure

does not have an established connection structure for the A3 call, so the `join_con` request has to hunt for it. Additionally, the accessibility of the connection is C, so the root is queried for permission.

Figure 5.12a and Figure 5.12b show the required operations and diagram respectively to the point where the `join_con` message finds an EST copy of the connection structure. TMP connection structures are created along the route from F to P2 in the intervening PSs P4 and P3. No bandwidth is allocated on the outgoing trip, because the requirements are unknown at that time.

Figure 5.13a, and Figure 5.13b show the operations and diagram for the `r_join_con` message. This message contains the information necessary to complete the connection structures in P3 and P4. The bandwidth is also allocated at this time. Since there is enough, the reply is successful. If there had been any problems (insufficient resources) then the reply would have been marked as a failure, and not all the PSs would have the full bandwidth allocated or completed connection structures. The reply does not return to CPE F yet, nor does data flow from the connection to CPE F at this time.

Since the accessibility of the connection was C, P4 sends out an `est_join` message and waits for its response before it replies to F. This message finds its way back to P2 along the same links created by the `join_con` message. When it reaches there, P2 causes a `join_req` message to be sent to the root (again, because the accessibility is C). This message follows the established connection back to the root, creating TMP links on the way. They are marked with identifiers different from those of the `est_join` message, because that message originated at P2. Finally this message is given to the root CPE. Figure 5.14a and Figure 5.14b show the

F→P4:	`join_con(A3,9)`

P4:	`create_con(A3)`
	`create_link(A3,F,P4_29)`
	`route(A)→P3`
	`create_link(A3,P3,P4_29)`

P4→P3:	`join_con(P4_29,A3,F)`

P3:	`create_con(A3)`
	`create_link(A3,P4,P4_29)`
	`route(A)→P2`
	`create_link(A3,P2,P4_29)`

P3→P2:	`join_con(P4_29,A3,F)`
P2:	`create_link(A3,P3,P4_29)`

Figure 5.12a: `Join_con` Message Finds Connection (Operations)

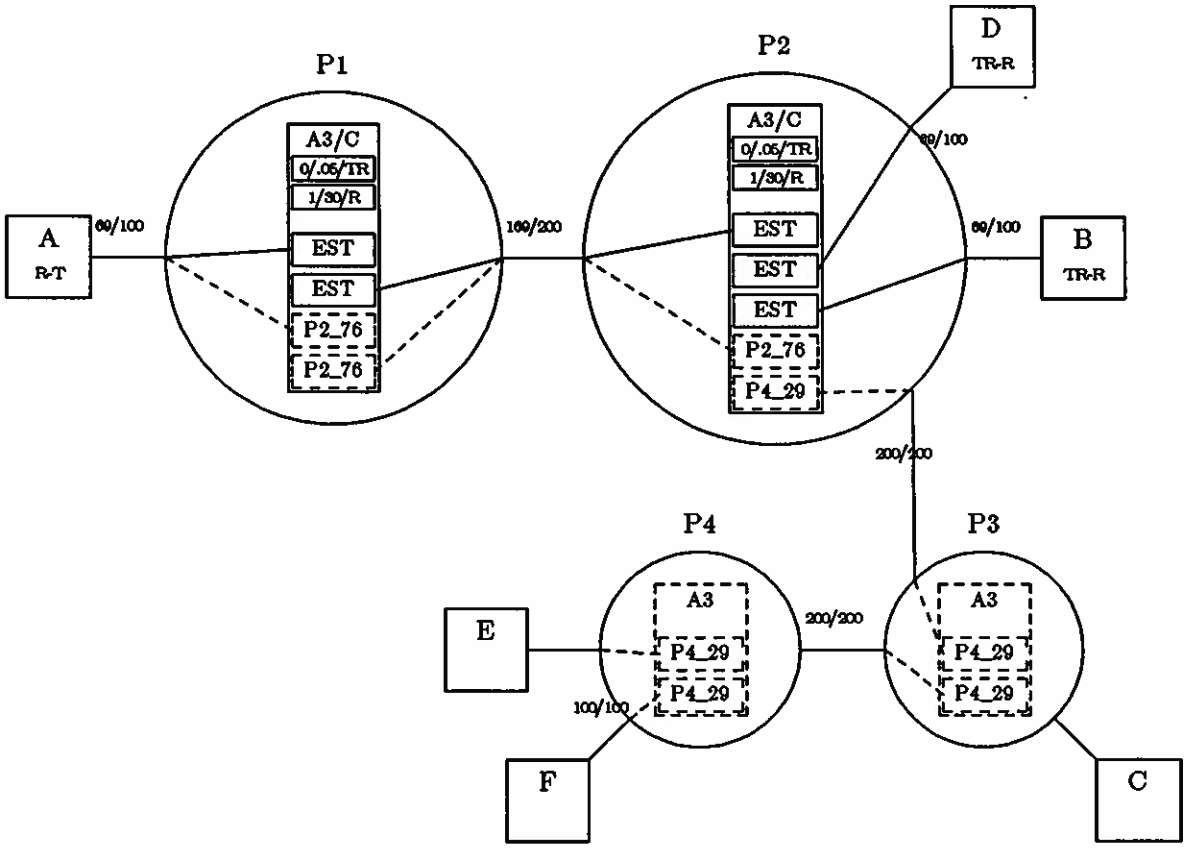


Figure 5.12b: `Join_con` Message Finds Connection (Diagram)

P2:	`alloc_bw(31,P3)`→`OK`
P2→P3:	`r_join_con(P4_29,A3,`*A3-Info*`)`

P3:	`alloc_bw(31,P2)`→`OK`
	`alloc_bw(31,P4)`→`OK`
	`mod_con(A3,`*A3-Info*`)`

P3→P4:	`r_join_con(P4_29,A3,`*A3-Info*`)`

P4:	`alloc_bw(31,P3)`→`OK`
	`alloc_bw(31,F)`→`OK`
	`mod_con(A3,`*A3-Info*`)`

Figure 5.13a: Originating PS Receives Reply (Operations)

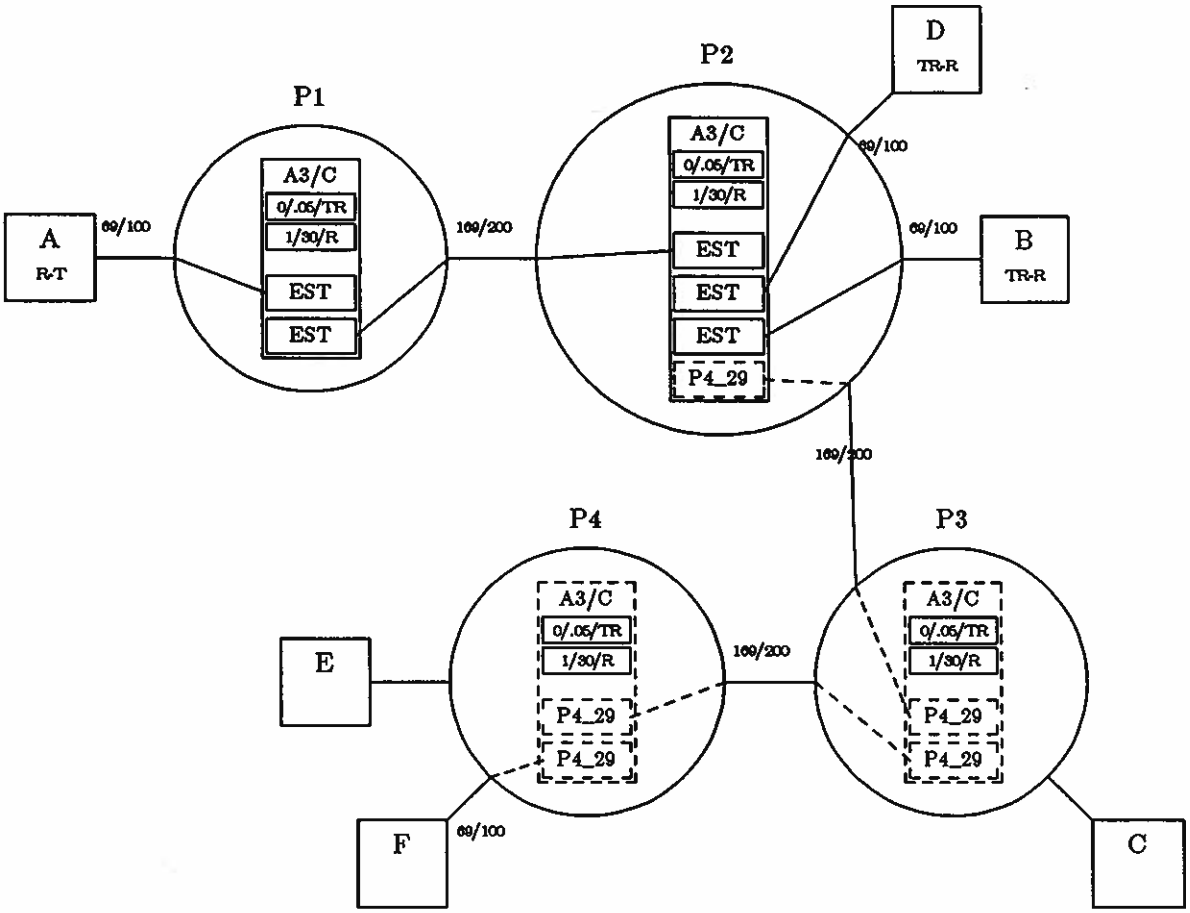


Figure 5.13b: Originating PS Receives Reply (Diagram)

operations and diagrams.

In this case, the root replies with permission to join the connection via the `r_join_req` message. This follows the P2_76 links back to P2, where it causes an `r_est_join` message to be sent back to P4. On its way back it marks all the A3 connections as established as well as the links. Upon return to P4 it causes the `r_join_con` message to be, at last, returned to F. The join request is thus completed. The operations to do this are shown in Figure 5.15a, and the final diagram in Figure 5.15b.

5.3.7. Endpoint is Deleted

Assume that C had been added to the connection by some means, but the root now wishes to delete it. The state of the connection after C had been added is shown in Figure 5.16a. After the operations shown in Figure 5.16b are completed, the state will be as it was depicted in

P4→P3:	`est_join(P4_29,A3,F)`
P3→P2:	`est_join(P4_29,A3,F)`
P2:	`create_link(A3,P1,P2_76)`
P2→P1:	`join_req(P2_76,A3,F)`
P1:	`create_link(A3,P2,P2_76)` `create_link(A3,A,P2_76)`
P1→A:	`join_req(A3,F)`

Figure 5.14a: Root is Queried (Operations)

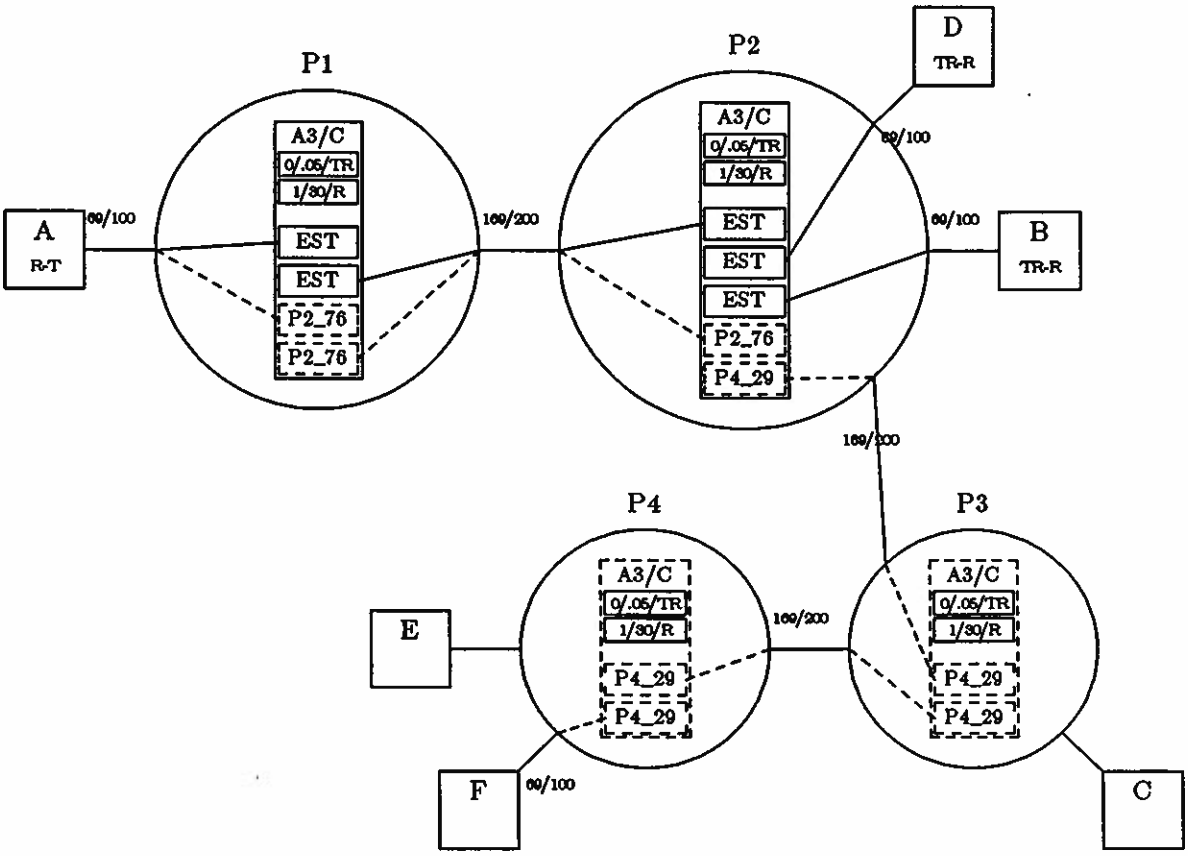


Figure 5.14b: Root is Queried (Diagram)

A→P1:	r_join_req(A3,F,OK)
P1:	del_link(A3,A,P2_76) del_link(A3,P2,P2_76)
P1→P2:	r_join_req(A3,F,OK)
P2:	del_link(A3,P1,P2_76) mod_link(A3,P1,P4_29,EST)
P2→P3:	r_est_join(P4_29,A3,F,OK)
P3:	mod_con(A3,EST) mod_link(A3,P2,P4_29,EST) mod_link(A3,P4,P4_29,EST)
P3→P4:	r_est_join(P4_29,A3,F,OK)
P4:	mod_con(A3,EST) mod_link(A3,P3,P4_29,EST) mod_link(A3,F,P4_29,EST)
P4→F:	r_join_con(A3,F,OK)

Figure 5.15a: Join is Completed (Operations)

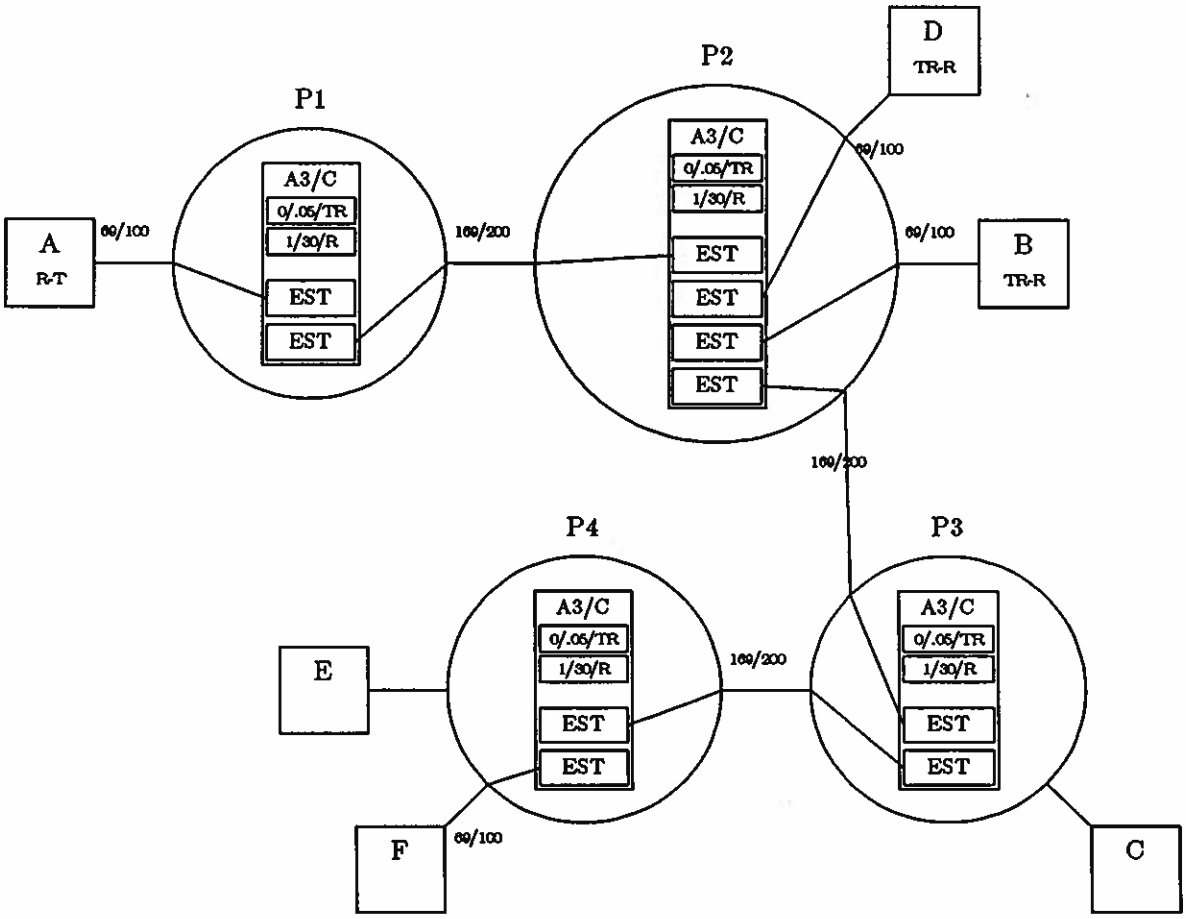


Figure 5.15b: Join is Completed (Diagram)

Figure 5.15b. These operations are very similar to those required for the `add_ep` message. TMP links are created from the root to the PS containing the link to C which is P3. At that point that link to C is deleted and the bandwidth freed. In our current example, the reply is merely sent back to the root without further action. If this had been the last EST link in P3, however, the connection structure would also have been deleted in P3.

5.3.8. Endpoint Drops Connection

An endpoint can remove itself from the connection with the `dropout` message. Doing so requires finding the first PS where the connection structure serves another CPE or more than two PSs, and then deleting the links and structures from that point back to the requesting CPE.

Figure 5.17a and Figure 5.17b show the operations and state after CPE F has issued a `dropout` message, and the first PS which serves other CPEs has been found. The connection

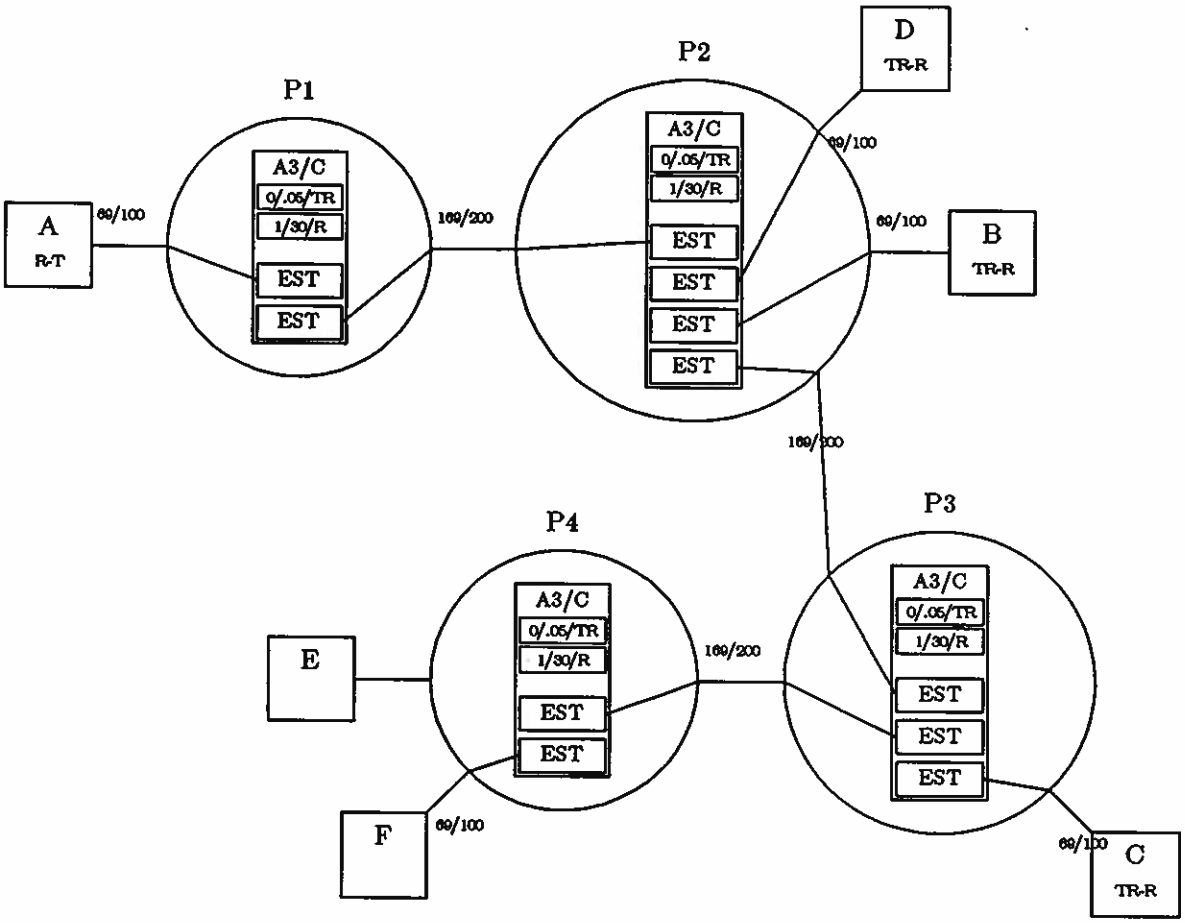


Figure 5.16a: C has been Added

```
A→P1:       del_ep(A3,C)

P1:         create_link(A3,A,P1_21)
            create_link(A3,P2,P1_21)

P1→P2:      del_ep(A3,C,P1_21)

P2:         create_link(A3,P1,P1_21)
            create_link(A3,P3,P1_21)

P2→P3:      del_ep(A3,C,P1_21)

P3:         create_link(A3,P2,P1_21)
            del_link(A3,P4)
            free_bw(31,C)
            del_link(A3,P2,P1_21)

P3→P2:      r_del_ep(A3,C,P1_21)

P2:         del_link(A3,P3,P1_21)
            del_link(A3,P1,P1_21)

P2→P1:      r_del_ep(A3,C,P1_21)

P2:         del_link(A3,P2,P1_21)
            del_link(A3,A,P1_21)

P1→A:       r_del_ep(A3,C)
```

Figure 5.16b: Endpoint is Deleted (Operations)

structures and links in P4 and P3 and the link from P2 to P3 are marked TMP, because they do not services any other CPEs besides F. The link from P2 to P3 is also marked TMP for this reason.

Figure 5.18 shows the operations necessary to complete this request. The state would be identical to that just after the video lecture was established. The completion is simply the reply message going back to F. The TMP connections structures and links are deleted and the bandwidth is deallocated.

5.3.9. Connection is Closed Figure 5.19 shows the operations necessary to close the entire connection. The new state diagram would merely be the four PSs with no connections shown. Instead of marking the connections TMP, they are marked CLOSING, so that other requests to join the connection fail immediately without searching further for established instances of the connection. The reply message causes all the bandwidth to be deallocated, and the connections structures and links to be deleted.

F→P4:	`dropout(A3)`

P4:	`mod_con(A3,TMP)`
	`mod_link(A3,F,TMP,P4_88)`
	`mod_link(A3,P3,TMP,P4_88)`

P4→P3:	`dropout(P4_88,A3,F)`

P4:	`mod_con(A3,TMP)`
	`mod_link(A3,P4,TMP,P4_88)`
	`mod_link(A3,P2,TMP,P4_88)`

P3→P2:	`dropout(P4_88,A3,F)`
P2:	`mod_link(A3,P3,TMP,P4_88)`

Figure 5.17a: Dropout Request Given (Operations)

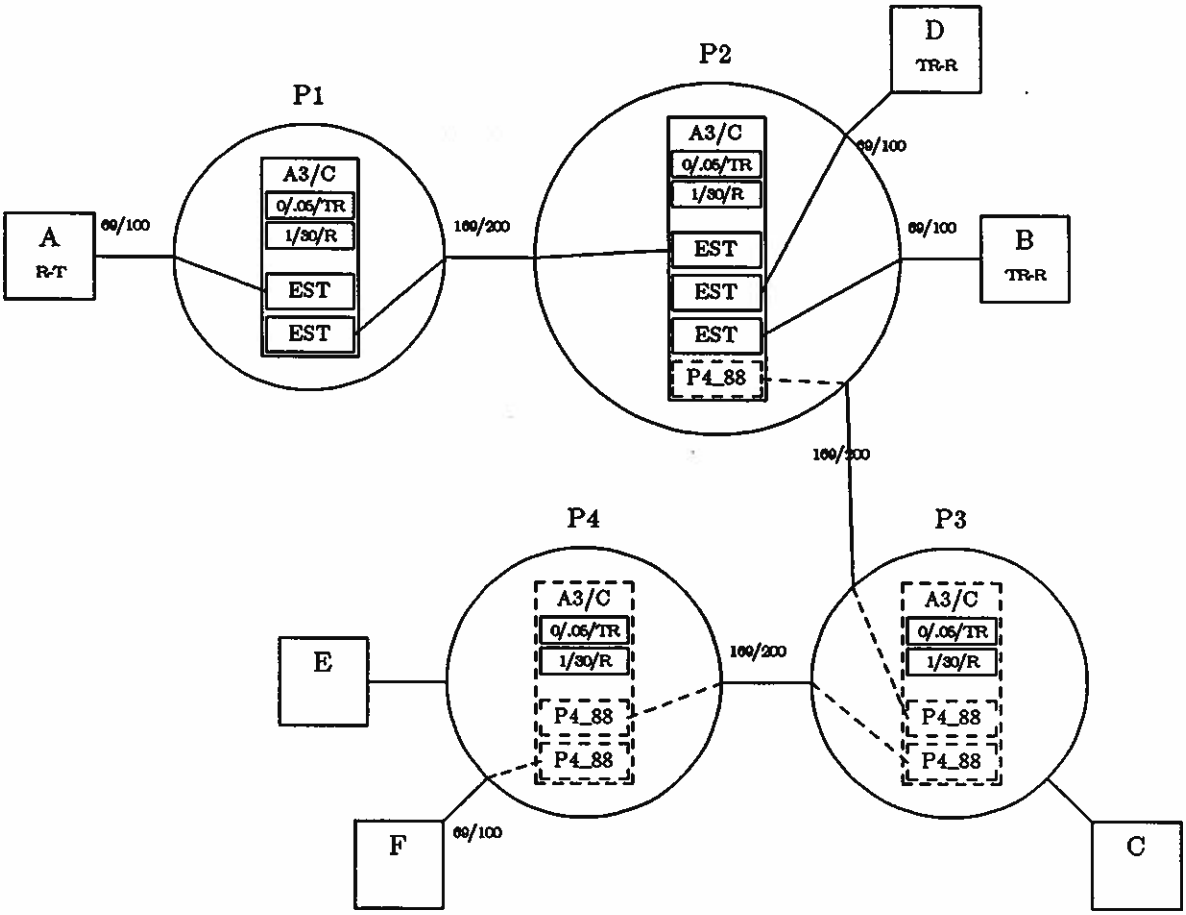


Figure 5.17b: Dropout Request Given (Diagram)

P2:	del_link(A3,P3,P4_88)
	free_bw(31,P3)
P2→P3:	r_dropout(P4_88,A3,F,OK)
P3:	del_link(A3,P2,P4_88)
	free_bw(31,P2)
	del_link(A3,P4,P4_88)
	free_bw(31,P4)
	del_con(A3)
P3→P4:	r_dropout(P4_88,A3,F,OK)
P4:	del_link(A3,P3,P4_88)
	free_bw(31,P3)
	del_link(A3,F,P4_88)
	free_bw(31,F)
	del_con(A3)
P4→F:	r_dropout(A3,OK)

Figure 5.18: Dropout Request Completed (Operations)

A→P1:	close_con(A3)
P1:	mod_link(A3,A,TMP,P1_99) mod_link(A3,P2,TMP,P1_99) mod_con(A3,CLOSING)
P1→P2:	close_con(P1_99,A3)
P2:	mod_link(A3,P1,TMP,P1_99) mod_link(A3,D,TMP,P1_99) mod_link(A3,B,TMP,P1_99) mod_con(A3,CLOSING) del_link(A3,D,P1_99) free_bw(31,D) del_link(A3,B,P1_99) free_bw(31,B) del_link(A3,P1,P1_99) free_bw(31,P1) del_con(A3)
P2→P1:	r_close_con(P1_99,A3,OK)
P1:	del_link(A3,P2,P1_99) free_bw(31,P2) del_link(A3,A,P1_99) free_bw(31,A) del_con(A3)
P1→A:	r_close_con(A3,OK)

Figure 5.19: Connection is Closed (Operations)

6. Level N+1 Specification: Customer Premisis Equipment (CPE)

This section is included to indicate how we feel certain customer operations might be performed given the BPN. In all cases it is assumed that a connection exists within the network, and the appropriate configuration (channels, accessibility, etc.) has been specified.

6.1. Simple telephone call

A telephone call can be broken into several steps which include: off-hook, digit collection, connection, ringing, answering, communication, and termination. Of these, only connection is provided by the BPN. The CPE would recognize the customer's device (telephone) going off-hook, would then collect the digits, and then might do some translation to determine the BPN address of the destination, which could actually be on the same CPE, so access to the BPN would not be required. The CPE in this case might be a digital PBX.

If BPN services were needed, the CPE would open a connection and contact the remote CPE. A connection like this should always succeed, and now the two CPEs would be communicating. The source CPE would indicate which device at the destination is desired, and the destination CPE would then attempt to contact it. All call progress information would be passed back to the source CPE using the BPN as the transmission mechanism. The BPN would see this CPE control information as data which is indistinguishable from that of the voice data passed once the telephone call is finally established.

Upon completion of the call, one CPE would signal the other, and the originating CPE would then tell the BPN to close the connection.

6.2. Voice-only conference calls

This would differ from the simple telephone call in that it requires a little more intelligence of the CPE. The CPE would have to sum and weight the signals from all the other CPEs in the conference call, so that the receiver was presented with a reasonable signal. Recall that the BPN would merely send all data from each CPE to each other CPE unmodified.

6.3. Broadcast television

Many of the features of the BPN make broadcast television simple. The broadcaster merely opens a connection, publishes the connection identifier, and begins to send the broadcast into the network. The accessibility would be A for commercial television and C for pay television.

Ideally the receiving CPEs would be televisions. The user would program the television so that certain channels corresponded to certain broadcast connections. When the channel is changed, the CPE drops out of the present connection and a new connection is sought. For efficiency purposes, old connections might linger, in case the user wanted to go back to previously viewed channels.

Instead of the CPEs being televisions they might be receivers at a local network affiliate which will rebroadcast the transmission by other means (e.g. cable). This is a more efficient use of the network in that these connections will be longer lasting and fewer in number.

6.4. Video lectures

Video lectures encompass all the features of the BPN, broadcast video, conferencing, and accessibility constraints. The CPE would, of course, have to be able to handle data at video rates. All encoding and compression would be done at the broadcasting CPE, with the receiving CPE doing the expansion and decoding. Again, the BPN would just provide the data path regardless of the data sent.

6.5. Electric meter reading

This, and other forms of telemetry, might require a datagram service of the BPN. This service has not been described in this paper, but is a part of the BPN as described in [4]. The utility would send out a read request datagram, and the meter would return a response. This would be more efficient than building a connection and then tearing it down for a relatively small amount of data.

6.6. Point-to-point file transfer

Here the CPE might be a computer. The BPN would correspond to the network layer and provide some features (namely connection) of the transport layer in the ISO OSI model. The CPE would have to provide the other transport layer functions such as an extended address space, end-to-end flow control, resequencing, and retransmission. It would also have to decide when it is best to build a connection or use the datagram service provided by the BPN.

7. Future Work

We plan to refine and extend the specifications presented in this paper. Most notably we need to define actual protocols for the operations specified. These upper level protocols should rely on robust lower level ones which should be specified as construction of a prototype network progresses, though they should be simulated before being actually implemented on hardware.

The routing concept needs to be refined, and the abstraction used here must be transformed into actual algorithms. A packet switch control protocol will then be required. Also the concept of bandwidth allocation and usage enforcement must be specified and algorithms for performing these functions designed and implemented.

Work at the higher levels will continue as we wish to prove the feasibility of new applications. We plan to specify or adapt higher level protocols which will allow us to follow each scenario presented in this paper, as well as others in areas such as customer name service and remote authorization. In the implementation of applications we hope to prove the adequacy of our lower level protocols, the functionality of the network, and provide examples for further application development by outside parties.

8. Conclusions

Specification of protocols for a network as large and as rich in features as the BPN is an enormous task. What we have tried to do here is concentrate on the connection management aspect at two of the many possible layers of abstraction. Other layers of abstraction require an equally, or more, detailed treatment, and, orthogonally, the techniques presented here need to be expanded to a more complete specification. In short, this is merely a beginning.

References

1. R. Bubenik, Performance Evaluation of a Broadcast Packet Switch, M.S. Thesis, Washington University Computer Science Dept., August 1985.

2. R. Callon, Proposal for a Connection-Oriented Internetwork Protocol, *ACM Computer Communication Review 13*, 3 (July 1983), 10-17, ACM.

3. J. S. Turner, Design of an Integrated Services Packet Network, WUCS-85-3, Washington University Department of Computer Science, March 1985.

4. J. S. Turner, Design of a Broadcast Packet Switching Network, WUCS-85-4, Washington University Department of Computer Science, March 1985.

5. J. S. Turner, Congestion Control in a Broadcast Packet Network, WUCS-86-??, Washington University Computer Science Dept., St. Louis, Missouri, ????? 1986.

6. J. S. Turner, Routing in a Broadcast Packet Network, WUCS-86-??, Washington University Computer Science Dept., St. Louis, Missouri, ????? 1986.