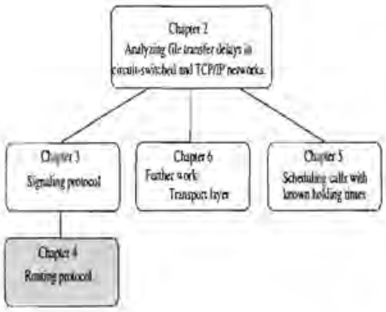
Chapter 4

Routing protocol

4.1 Introduction


Chapter 2
Analyzing file transfer delays in circuit-switched and TCP/IP networks.
Chapter 3
Signaling protocol
Chapter 6
Futher work:
Transport layer
Chapter 5
Scheduling calls with known holding times
Chapter 4
Routing protocol

An accompanying routing protocol is required for the signalling protocol presented in Chapter 3. Recall from Section 3.4 that we selected to use hop-by-hop routing and route precomputation for the signalling protocol. This means that when a request for a connection arrives at a node, the routing table should contain all the information required to produce a next-hop node address for any address reachable through itself.

The routing table produced by the routing protocol should also contain enough information to be able to accommodate connection setup in a heterogenous network. The "next-hop node" entry, J, in the routing table of node I may be a node that is not *directly* connected through a physical or logical link to node I. In order for node J to be considered a "next-hop node" by node I, node J *must* operate at the same or lower rate as node I. As a result, on a physical path from node I to the next-hop node J there may be one or more intervening nodes that are not regarded as next-hop nodes. If the "next-hop node" entry in a routing table indicates a node that is not an immediate neighbour of node I (physically or logically), a second entry is needed indicating which immediate neighbour should be used to reach the "next-hop node". Section 4.2 starts by introducing the addressing scheme that will be used by the routing protocol, Section 4.3 continues with the design decisions, and Section 4.4 describes the routing protocol.

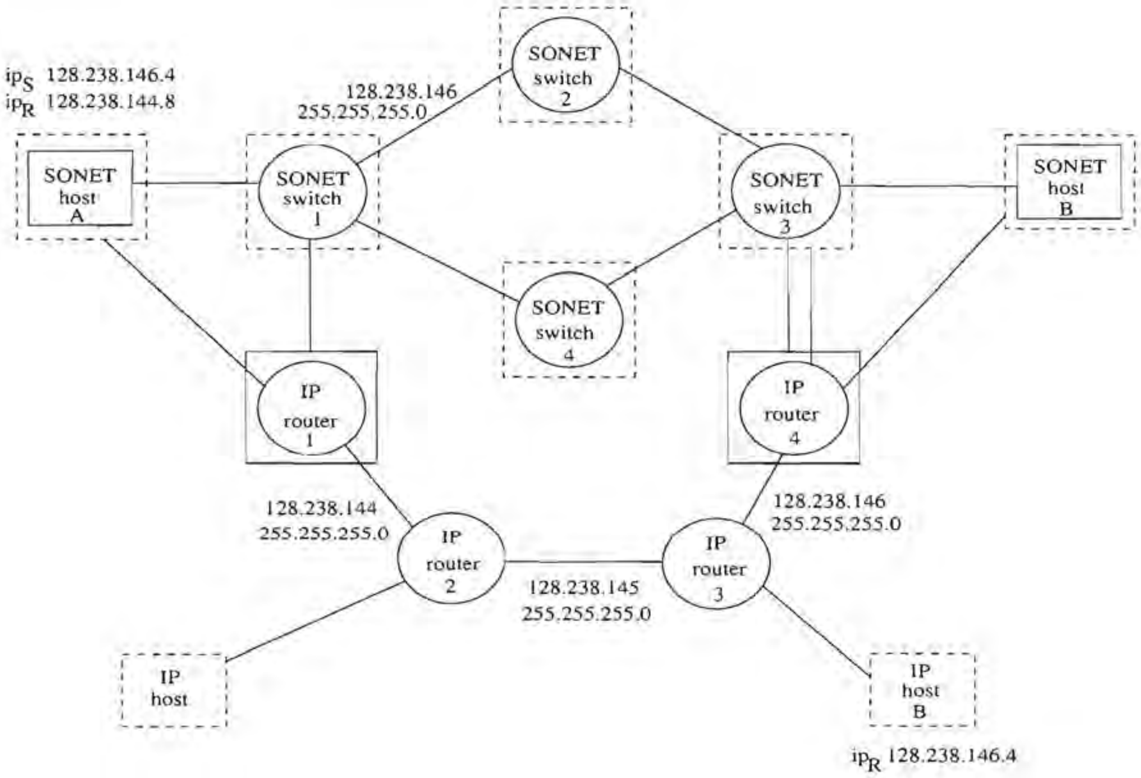


Figure 4.1: Routing network

4.2 Addressing

For an efficient, low overhead routing protocol, the need exists for an addressing scheme that allows for summarization. Address summarization refers to the use of address prefixes to represent a collection of end host addresses that begin with the same prefix. Only the address prefixes need to be distributed between nodes participating in the routing protocol and represent the reachability of all end hosts. This section presents an addressing scheme for use in SONET networks. It is proposed that a second set of IP addresses be used to identify all nodes forming part of the SONET network, an addressing scheme that lends itself to address summarization without reducing the number of IP addresses available to the IP network

A benefit of using IP addresses is that recent work on route lookups in hardware, [8] and [16]-[18], can be leveraged for the first action of route determination needed in call setup.

Figure 4.1 shows an example network in which the signalling protocol presented in Chapter 3 will be implemented. In the figure all IP hosts are indicated by a dashed rectangle and all SONET hosts are indicated by a solid rectangle. This example network, where only some interconnections are indicated, shows the co-existence of the IP and SONET networks. All SONET hosts and SONET switches are IP hosts, and some IP routers are hosts of the SONET network. It is assumed that all SONET hosts and switches will have IP interfaces, for example in the form of Ethernet interfaces. For example, in Figure 4.1 the link between SONET host A and SONET switch 1 is a SONET interface, while the link between SONET host A and IP router 1 could be an Ethernet interface. IP routers are hosts of the SONET network in the Internet today where there are provisioned dedicated channels set up between routers interconnecting large sites.

With the above network design, the decision to make use of IP addresses for signalling and routing in the SONET network becomes plausible. Although every SONET switch and host already has an IP address, the usage of these addresses will introduce an immense strain on the routing protocol by increasing the sizes of the routing databases considerably. The reason for this is that in the construction of the network, it cannot be guaranteed that two neighbouring (connected by a point-to-point link) SONET nodes will be part of the same IP subnet. As a consequence, each SONET node has to be identified by its complete 32 bit IP address in the routing protocol, preventing the usage of address summarization.

For the routing protocols to make use of IP addresses while retaining the capability of address summarization that is provided with the use of subnet addressing, two preconditions are met. Firstly, in the network presented in Figure 4.1, it can be seen that one node of one network type can only be a host of the other network (as opposed to a router or switch). For example, an IP node (router or end host) can only be an end host of the SONET network, not a switch. Similarly, a SONET node can only be an end host of the IP network, not a router. Secondly, an IP address is assigned to each interface, IP and SONET, forming two distinct sets of IP addresses, which do not need to be disjoint.

If ip_R indicates an IP address of an IP interface, and ip_S indicates an IP address assigned to a SONET interface, Figure 4.1 presents a possible address assignment to SONET host A. Host A has an ip_R of 128.238.144.8 assigned to its interface that is connected to IP router 1. It also has the ip_S address of 128.238.146.4 assigned to the interface connected to SONET switch 1. Figure 4.1 also shows that SONET host A and IP host B now share an IP address. Although it is the same IP address, there shall never be any conflicts due to the context in which the addresses are used. Any IP address used in an IP header will always be an ip_R address and any IP address used in the called party address field of a SONET circuit signalling message will be an ip_S address. All signalling and routing messages of the SONET network will be carried as IP packets with ip_R addresses in the headers and hence will reach the appropriate SONET destinations because they are hosts in the IP network. The SETUP message (without the IP header) contains two IP addresses that identify the source and destination nodes of the SONET network between which a connection has to be set up. These two IP addresses are always ip_S addresses. It is to find a route between this source and destination pair that a routing protocol is needed. So, the IP addresses used for the actual signalling will never be present in an IP header, only in the SETUP message. Because these addresses will never be used in the same context, it allows the SONET network to have nodes identified by IP addresses already used in the IP network.

In a network as depicted in Figure 4.1, there will always be two routing protocols running concurrently. The IP routers exchange information about reachability of nodes with ip_R addresses, and the SONET switches exchange information about reachability of nodes with ip_S addresses. These two routing protocols never exchange routing information between each other.

It is now possible to assign IP addresses to SONET interfaces independent of the IP address assigned to their IP interfaces, allowing for a SONET switch and all its neighbours to share a subnet address. Thus allowing for address summarization.

All data tables discussed in the signalling protocol description shall contain ip_S addresses. The tables concerned are the *Routing table*, the *Connectivity table* and the *State table*. This introduces the need for a new table that provides a mapping between ip_R and ip_S addresses of which Table 4.1 is an example. While participating in connection setup, a switch will always send a signalling message to a neighbour. Note that this might not be an immediate neighbour (connected via a point-to-point link) in a heterogenous network - it could also be a node to which a connection has already been set up - a *logical neighbour*. In a homogeneous network, this table can be set up by the administrator due to the fact that signalling messages will only be sent to immediate neighbours (two nodes connected with a point-to-point link). In a heterogenous network, the initial information in the *ip_S to ip_R mapping table* (which is a mapping of the addresses of a node's immediate neighbours) is entered by the administrator. To be able to handle connection setup in a heterogenous network the signalling protocol should be able to send signalling messages to logical neighbours. For this functionality, the signalling protocol needs to add one more step during connection setup. Once a connection has been set up between two switches,

ip_S	ip_R
128.238.146.4	128.238.144.8

Table 4.1: *ip_S to ip_R mapping table* kept by SONET switch 1

the source switch needs to update the *ip_S to ip_R mapping table* to include the ip_S to ip_R mapping of the destination. This mapping can be inserted upon the receipt of a SETUP messages, and removed upon receipt of a RELEASE CONFIRM message.

The signalling protocol now requires an extra step before a signalling message is sent: each time a signalling message needs to be sent, the address mapping table has to be queried to determine the IP address that needs to be placed in the IP header. An example step when SONET switch 1 receives a request for a connection from SONET host A to SONET host B. Switch 1 first queries its routing table to find the ip_S address of the next hop to which the SETUP message should be sent. Then after CAC and switch fabric configuration it queries the *ip_S to ip_R mapping table* to find the ip_R address of the next hop node. It constructs a SETUP message and requests the IP network to deliver it to the SONET node with IP address ip_R.

4.3 Design decisions

The usage of IP addresses by all the nodes of the SONET network allows for the use of an existing routing protocol, for example OSPF [19]. Using a routing protocol like OSPF will only require two changes: first, before any routing messages are sent on the IP network, the *ip_S to ip_R mapping table* has to be queried to find the ip_R address. Second, the construction of the routing table has to be changed to enable routing in heterogenous networks. OSPF even contains rudimentary support for Quality of Service (QoS), which allows the exchange of node state (real-time) parameters, for example the current available bandwidth of outgoing interfaces.

Although an existing IP routing protocol may be used, this discussion shall continue with the design of a lightweight routing protocol designed specifically to be used in conjunction with the signalling protocol described in Chapter 3.

The presence of the IP network will be ignored in the rest of the routing protocol description. The arguments for the creation of the hop-by-hop, link-state routing protocol that creates a routing table by constructing a shortest path tree is described next.

Section 3.4 briefly mentioned the decision that the options of **route precomputation** and **hop-by-hop** routing (as opposed to source routing) will be used in the routing protocol. Using SONET it is possible to make full use of route precomputation due to the granularity of the multiplexing rates. As depicted in Table 3.3, next-hop entries are only kept for the SONET rates of OC1, OC3, OC12, OC48 and OC192. This simplifies the connection setup procedure in that the route determination step is just a table lookup. There are two reasons for the decision of hop-by-hop routing. Firstly, by using hop-by-hop routing the parsing of the SETUP message is simplified at each hop allowing for hardware implementation. Secondly, when source routing is used it is possible that conditions change as connection setup proceeds from a given source node toward a destination, and by the time the setup request reaches an intermediate node, the selected source route is no longer available, in which case crankbacks are needed. Crankbacks require the management of more state information, which leads to increased hardware complexity.

The routing protocol should be executed in parallel by all the SONET switches, without segmenting the network. This allows all the nodes to have an identical copy of the routing database. The reason that the current routing protocol proposal requires a flat network structure (no segmentation) is for simplicity. The moment a network is segmented, the different "areas" can receive summarized information regarding the real time parameters in the other areas. Because the information is summarized, some mechanism similar to Generic CAC (GCAC) from PNNI needs to be applied to a connection request before it is passed on to the next hop (together with the CAC performed by the node itself). GCAC allows a node to predict the outcome of the actual CAC performed at another switching system given that node's advertised additive link metrics. Performing GCAC will add complexity to the signalling protocol, and will heighten the risk that the real CAC performed by a switch further along the path (possibly in another area) may fail. This failure will introduce crankback. A flat network structure avoids this problem.

Routing databases tend to be very large. Using a routing protocol to distribute real time parameters, for example the available bandwidth of an interface, requires the protocol to always have up to date information in the database and to respond quickly to any changes in the database. A distance-vector routing protocol such as RIP [20] would be inefficient when used in large networks (large routing databases). A distance-vector algorithm, such as the Bellman-Ford algorithm used by RIP, does not perform well when there is a large database and a topology change occurs or a change in a node's real time parameters occurs. This is because a change in the network results in a convergence period which might not be fast enough for the routing protocol to be usable.

In distance-vector algorithms, each node keeps a table with an entry for every possible destination reachable though itself. The entry indicates the distance to the destination and the next-hop node to the destination. The distances are continually compared and updated as routing update messages are received by a node. Once a topology change occurs (for example a link goes down), and the changes to the distances are distributes through the network, the routing tables will not immediately see the change because some nodes will still consider the link as active and reflect that in their advertisements. This is the cause of the convergence period that is the time for a node's routing table to reflect the correct distance values to the affected destinations.

It is thus proposed to make use of a **link state** routing algorithm. In this algorithm each node has an identical routing database containing the identity (address) and connectivity of each node in the network from which a routing table is constructed. The routing table entries are calculated by constructing a shortest path tree. By using a link state routing algorithm, any changes in the network topology or real-time parameters of nodes in the network will be received by all the nodes executing the routing protocol very quickly, thus enabling all the nodes to have the most up to date information regarding topology and real time parameters. With this information the routing protocol is able to make more accurate routing decisions than a distance-vector routing protocol.

Information distributed by the switching systems participating in the routing protocol should contain enough detail about their respective *identity* (reachability) and *capability*. Information concerning individual end hosts' *capability* (for example an end host's interface rate) cannot be distributed due to the usage of address summarization. This information (*identity* and *capability*) can also be described as *nodal information* and in this implementation it describes the node's address (the ip_S address) and the cross connect rate of the switching system. Together with the *nodal information*, the protocol also distributes *topological information*, which is information regarding the links between the switching systems. The *topological information* is a non-negative cost assigned to an outgoing interface based upon the amount of available bandwidth at the interface: the higher the available bandwidth of an interface, the lower the cost value. A lower cost value thus indicates a higher probability for this interface to be chosen.

At initialization, a node advertises that it has an available bandwidth equal to the interface rate (resulting in a low cost value). It is proposed that the distribution of subsequent topological information be based upon threshold computations. That is the node participating in the routing protocol should not send a routing update message every time a connection is admitted or released. Only when the available bandwidth reaches a certain threshold should it send out a routing update message containing a higher cost value to indicate a change in its available bandwidth.

As with the signalling protocol, the routing protocol also makes use of the IP network to distribute routing database updates. Considering the delay involved in the connectionless network, the usage real-time parameters might introduce some connection setup failures. For example, if node I passes a connection request to node I+1 based on the information in its routing database that states that node I+1 does have enough bandwidth available, and at the same time , node I+1 sends out a routing update message that it does not have enough bandwidth available - the connection request will fail. For this reason multiple next hop entries are kept in the routing table. If the connection request fails, the setup request is reconstructed and sent to the second (or third) option for a next hop node.

4.4 Protocol description

The routing protocol will be described through its four components. All the switching systems in the network have an identical **routing database** that describes the *nodal* and *topological information* of each node in the network. This database allows each node to construct a **directed graph** representing the network. From this directed graph, each switching system constructs the **shortest path tree** with itself as the root, and using this tree it is now possible to compute the **routing table** entries.

An example network is shown in Figure 4.2. This example, and all the examples in this chapter will refer to the switching systems by their names (SwitchA, SwitchB, etc.). Note that this representation is used to simplify the explanations, an implementation would use the IP (ip_S) addresses. Each directed line indicates a unidirectional connection between two switching systems. A cost is associated with each outgoing interface. The cross connect rates are also depicted because these values play an important role in the construction of the routing table.

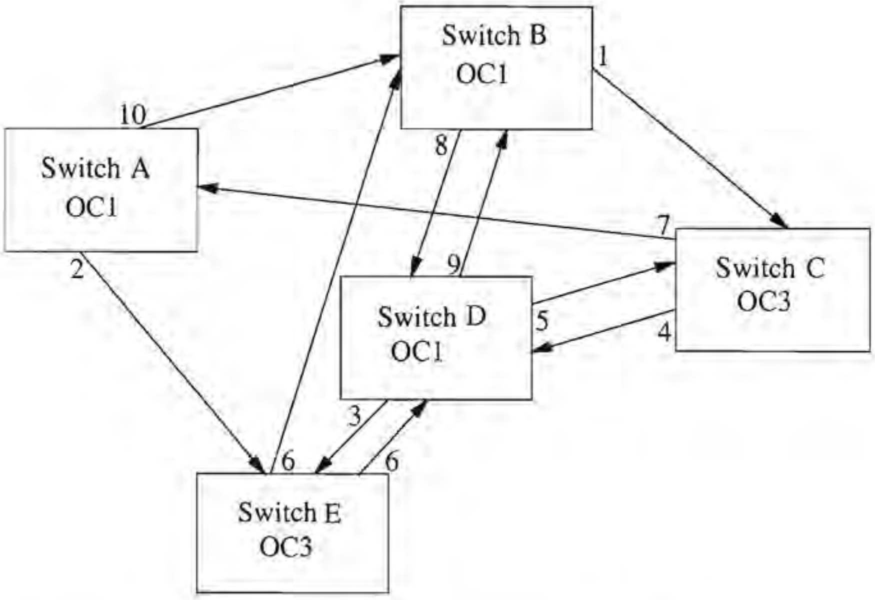


Figure 4.2: Example network configuration for routing protocol

Next hop node	Interface number	Total bandwidth	Available bandwidth	Cost

Table 4.2: Available bandwidth table including cost

4.4.1 Routing database and directed graph

For this routing protocol, each switching system needs to know to which other nodes it is directly connected. This information is currently stored in the *Connectivity* and *Available bandwidth* tables used by the signalling protocol. All connections will be unidirectional, so we are only interested in the succeeding, directly connected nodes. This information is stored in the *Available bandwidth* table. The cost associated with an interface is programmed into the *Available bandwidth* table, Table 3.4, of which Table 4.2 is a copy. The cost is a decreasing function of the available bandwidth of an interface - the higher the available bandwidth, the lower the cost.

For each node n (including itself) in the network, the routing database of each switching system in the network contains information indicated by the first column of Table 4.3.

- **Node id** refers to the ip_S address of node n.
- The **sequence number** characterizes the last routing update concerning node n.

Node id	SwitchD
Seq	x
Connectivity	SwitchB, SwitchC, SwitchE
Cost	9, 5, 3
Cross connect rate	OC1

Table 4.3: Routing database entry for node D from network depicted in Figure 4.2

This number is used to determine if a routing database update received from any node about node n does in fact contain new information, a higher sequence number indicates more recent information. The usage of the sequence number will receive more attention later.

- **Connectivity** indicates all the nodes directly connected to node n. That is, all the nodes of which node n is the preceding node in a unidirectional point-to-point physical connection. Nodes are identified by a combination of ip_S address and netmask. For example, an ip_S address of 128.238.144.0 and netmask of 255.255.255.0 indicates that all the hosts on the 128.238.144 network are directly connected to (and consequently reachable through) node n.

- **Cost** refers to the costs associated with the interfaces connecting this node (n) to other nodes. Each entry is an indication of the available bandwidth on the interface connecting node n with a node indicated by the *Connectivity* information - the lower the cost, the higher the available bandwidth.

- **Cross connect rate** is given if node n is a switching system.

With this information in the routing database it is straightforward to construct the directed graph.

If x is the highest sequence number of a routing update received concerning SwitchD, an example of an entry in the routing database for SwitchD is shown in Table 4.3. This entry will be identical in all the nodes in the network.

The cost assigned to interfaces which connect a switch to its end hosts should typically be equal. For example, if all the hosts on the 128.238.144 network are reachable through SwitchD, it shall advertise its connectivity with an ip_S address of 128.238.144.0, netmask of 255.255.255.0 and only one cost value. If there are different costs assigned to interfaces connecting hosts (or switches) that have addresses that can be summarized, the highest cost of all the interfaces will be used in all routing databases.

To limit the usage of network bandwidth a node initiates database exchange with its neighbour only on three occasions:

- At initialization.
- After any change occurs in its routing database. This includes any changes in its own connectivity, an increase over the threshold of the available bandwidth, a decrease below the threshold of the available bandwidth, or a routing update received from another node.
- After a very large time interval. It is assumed that the above two database exchange instances will guarantee database synchronization most of the time, but it is also necessary to have some error prevention mechanism, which is provided by this database exchange condition.

When node n's routing database entry changes, it should only send the new database entry to its neighbours, after which it is distributed through the network. In the other two database exchange occasions, the whole routing database is distributed among the nodes.

Routing databases are not flooded through the network. Each node shall send its routing database update message(s) to all of its neighbours that are reachable through itself (the "next-hop nodes" found in the *Available bandwidth* table) and that are participating in the routing protocol. For example, in Figure 4.2 SwitchA will send its routing database update message(s) to SwitchB and SwitchE, but these latter switches will not relay the messages on to SwitchC or SwitchD.

Only one message type is needed for this routing protocol. For each node n in the network about which a node has information in its routing database, it shall construct a message as presented in Table 4.4. It is possible to include more than one full message in an IP datagram.

The sequence number is only changed when the information it refers to has changed, and it is also only changed by the node to which the information refers. When a node sends its database to its neighbours after the large timeout it uses the same sequence number associated with its information in the database. Continuing the example this means that until a change occurs in SwitchD's information, the sequence number will continue to be x.

Information element	Size (bits)
Source node	32
Sequence number	31
Flag	1
Cross connect rate	4
Number of destinations (*nr. dest*)	12
Destination (1)	32
Netmask (1)	32
Cost (1)	16
...	
Destination (*nr. dest*)	32
Netmask (*nr. dest*)	32
Cost (*nr. dest*)	16

Table 4.4: Routing update message

The sequence number is a 31 bit unsigned integer. Each time there is a change in a node's connectivity it shall increment the sequence number associated with its information in the database and initiate the routing database update with its neighbours. Thus, the larger the sequence number, the more recent the database update. The problem with using a sequence number is deciding what to do when the sequence number has to be incremented past $2^{31} - 1$. A solution for this problem is to have an extra bit (*Flag*) in the routing update message. When a sequence number reaches $2^{31} - 1$ a node can reset the sequence number to 0 and set the extra bit to 1 indicating that this is a new routing update. All nodes in the network should now treat this message as if a message with a higher sequence number has arrived. The extra bit in the routing message should be set to 0 on all other occasions.

Using 12 bits to indicate the number of destinations allows any switching node to advertise 4095 different destinations reachable through itself. This is a worst case. The number of routes advertised by any node should typically be very low.

The 4 bit cross connect rate value is the cross connect rate of a switching system or the interface rate of an end host (although an end host should not typically participate in the routing protocol). The possible values of this information element is presented in

Cross connect/Interface rate	Value in routing message
OC1	0001
OC3	0010
OC12	0011
OC48	0100
OC192	0101
OC768	0110

Table 4.5: Possible values for the cross connect rate of a switch

		From				
		Switch A	Switch B	Switch C	Switch D	Switch E
To	Switch A			7		
	Switch B	10			9	6
	Switch C		1		5	
	Switch D		8	4		6
	Switch E	2			3	

Table 4.6: Directed graph

Table 4.5.

The directed graph for this network is represented in Table 4.6 and is identical at each node participating in the routing protocol.

4.4.2 Constructing the shortest path tree

The shortest path tree is constructed from the directed graph using the Dijkstra algorithm. The construction of the shortest path tree will be described next. The directed graph contains enough information for a switching system to construct the shortest path tree using itself as the root. We deviate from common graph terminology in this explanation by continuing to use terms from previous explanations. That is, instead of the normal usage of "vertex" or "edge" the words "node" and "link" will be used to explain the algorithm. The algorithm assumes the availability of a function w such that $w(u, v)$ returns the cost of the interface from **u** to **v** as retrieved from the directed graph. The algorithm proceeds as follows:

1. Create the four data structures. **V** is a list (or array) of all the nodes in the network (identified by ip_S address and netmask combinations) whose shortest path have yet to be found, **S** is a list (or array) of nodes whose shortest paths from the root have already been determined, **d** is a list (or array) of the current cost estimate along the shortest path to each node, and **pred** is a list (or array) of of each node's predecessor on its shortest path to the root.

2. Set **S** to empty.

3. Denote the cost to node u by $d[u]$. Set $d[u] = 0$ if u is the root and $d[u] = \infty$ otherwise.

4. Denote the predecessor of each node u by $pred[u]$. Set $pred[u] = u$ if u is the root and $pred[u] = NIL$ otherwise.

5. While there are still nodes in **V**:

   (a) Sort the nodes in **V** according to their current values in **d**.

   (b) Remove the closest node **u** from **V**.

   (c) Add **u** to **S**.

   (d) For each node, **v**, that is adjacent to **u** in the directed graph.

      i. if $d[v] > d[u] + w(u,v)$ then:

         A. $d[v] = d[u] + w(u,v)$

         B. $pred[v] = u$

Using the Dijkstra's algorithm the shortest path tree computed by SwitchB from Figure 4.2 is depicted in Figure 4.3. The shortest path from the root node to a node, n, can be found by traversing the predecessor links from n until the root node is encountered. If the node, n, has a NIL predecessor link, then there is no path from x to the root node. The predecessor list for SwitchB is presented in Table 4.7. Traversing the predecessor list to find the shortest path to SwitchE results in the path SwitchB-SwitchC-SwitchD-SwitchE. The first node in the shortest path will be used in the construction of the routing table. This will only produce one next-hop, in Chapter 3 the routing table (Table 3.3) contains three next-hop entries. To compute the additional two optional next hops that will be placed in the routing table (as shown in Table 3.3), the root node can construct two additional shortest path trees. After constructing the first shortest path tree, all links used

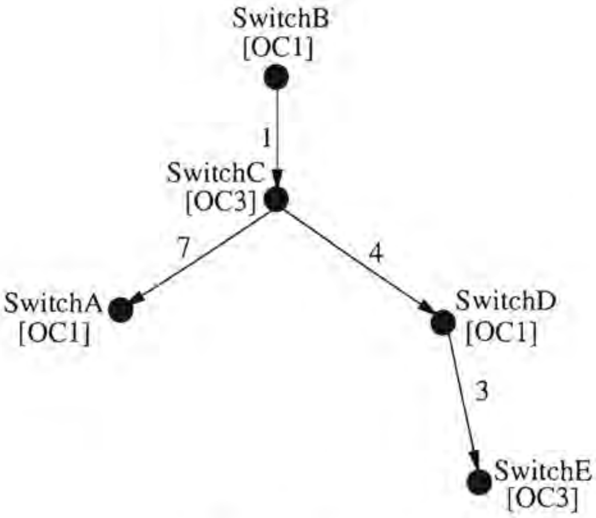


Figure 4.3: Shortest path tree constructed by node B

Node	Predecessor
SwitchA	SwitchC
SwitchB	SwitchB
SwitchC	SwitchB
SwitchD	SwitchC
SwitchE	SwitchD

Table 4.7: Predecessor list for SwitchB

in the tree are removed from the directed graph. The new directed graph is then used to construct a shortest path tree from which the second routing table entries are made. The same procedure is followed to compute the third optional next hop entries in the routing table.

4.4.3 Routing table

The shortest path tree of a node has the information for a node to decide on the next hop node to the destination. The next hop to a destination can be found by traversing the predecessor list of the shortest path tree. In a homogeneous network, the routing table can be constructed by each node *root* by considering each node n in turn. The IP address of the first hop in the shortest path from node *root* to node n becomes the entry of node n in the routing table of node *root*.

So far in the discussion it has been assumed that when a request for a connection

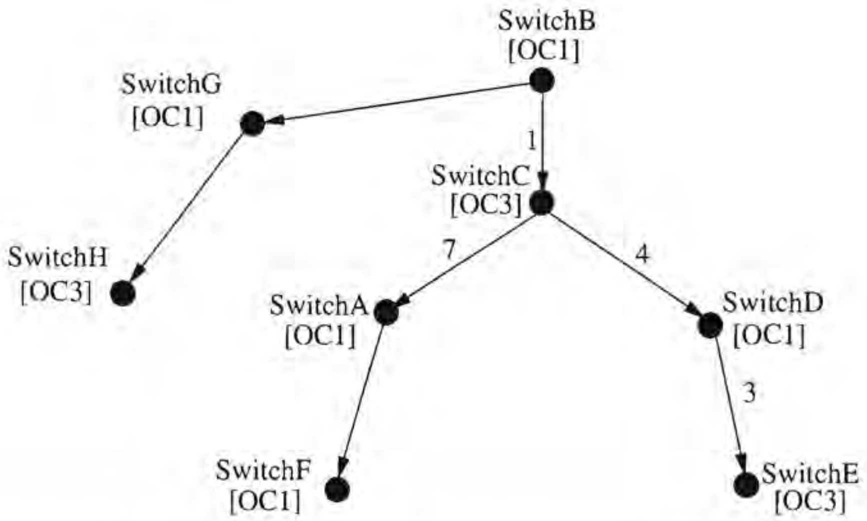


Figure 4.4: Shortest path tree constructed by node B, with extra node

arrives, the first step would be to do a route lookup in the routing table to find the next-hop node to the destination. It is also possible to first search the *Available bandwidth* table to find out if the destination is directly connected before the lookup is done in the routing table. This discussion will now continue the design where the routing table is always queried first. Thus if the destination is directly connected, the "next-hop node" entry will contain the IP address of the destination.

For this explanation, the shortest path tree in Figure 4.3 has been changed to Figure 4.4.

For this routing table to be useful in a heterogenous network, the construction of the routing table by a node (*root*) can be described as follows:

Consider the shortest path P from *root* to destination node n.

- If the next hop node *next* has the same or lower cross connect rate as node *root*, make an entry in the routing table under the section with the same cross connect rate as *next*.
- If the next hop node *next* has a higher cross connect rate than node *root*:
    - Find the closest *neighbour* node on path P that has the same or lower cross connect rate as node *root*.
        - If the *neighbour* node is found and is not the same node as n:
            - Place this node (*neighbour*) in the routing table as the "next-hop node"

used to reach node n. This entry is made at the crossconnect rate of *neighbour*.

  · Search the path from *root* to *neighbour* for the node with the highest cross connect rate (*highrate*).

  · Place *next* in the routing table as the next-hop node to reach *neighbour*. This entry is made at the cross connect rate section *highrate*.

- ∗ If the *neighbour* node turns out to be n or if no *neighbour* node is found:

  · Search for the node with the highest cross connect on path P

  · Place the entry for the next hop node *next* in the routing table at this cross connect rate section.

Due to the granularity of the SONET network, each entry made using the above description can be inserted at all the rates higher than its original position, if there are outgoing interfaces that operate at these or higher rates. It is impossible for a switch to connect to another node at a rate lower than its crossconnect rate, so all entries made to this respect has to be removed after the previous step has been completed.

When SwitchB constructs the routing table according to the shortest path tree depicted in Figure 4.4, it will proceed as follows. The first node it considers is SwitchC, which has a higher cross connect rate than SwitchB. As explained above, SwitchB shall search the path to SwitchC for a *neighbour*, there is no neighbour on this path, and the highest cross connect rate is OC3 - so an entry for SwitchC will be made in the routing table block under the rate OC3. Next, consider node SwitchF. Again, the next hop node (SwitchC) has a higher cross connect rate than SwitchB. Searching the path for a node that has the same or lower cross connect rate than SwitchB reveals that SwitchA is the *neighbour*, an entry is made in the routing table indicating that SwitchA is the "next-hop node" to SwitchF. Searching for the node with the highest cross connect rate on the path to SwitchA produces SwitchC. So, an entry in the OC3 section will indicate that SwitchC is the "next-hop node" for SwitchA. The resulting routing table for SwitchB in Figure 4.4 is presented in Table 4.8.

Another example network is depicted in Figure 4.5. This example network contains a path that is similar to the path describing the signalling protocol's operation in a heterogenous network given in Chapter 3. The path from Switch3 to Switch6 discussed in Section 3.9.1 can be found in Figure 4.5. The corresponding routing table, before removing the entries that Switch3 cannot accommodate due to its OC12 crossconnect rate, is given

Destination node address	Data rate OC1			Data rates OC3 to OC192		
	Next hop option 1	Next hop option 2	Next hop option 3	Next hop option 1	Next hop option 2	Next hop option 3
SwitchA				SwitchC		
SwitchC				SwitchC		
SwitchD				SwitchC		
SwitchE	SwitchD			SwitchD		
SwitchF	SwitchA			SwitchA		
SwitchG	SwitchG			SwitchG		
SwitchH	SwitchG			SwitchG		

Table 4.8: Routing table for node B

in Table 4.9.

4.5 Future work

The segmentation of the network is a functionality that is required in a routing protocol when it operates in a large network. Without network segmentation the routing databases tend to grow very large. Despite route precomputation being used, this step (route precomputation) could still consume a lot of resources, even if nodes only exchange a small number of routing updates. The first step in adding this functionality would be to specify the usage of IP addresses in the network segments, and secondly, the creation of the routing table has to be re-examined. The costs advertised between network segments will contain summaries of the cost values in the particular network segment - the next hop entry in the routing table should now contain the next hop to a network path that is most likely to be able to handle the connection based on more relaxed parameters that basic CAC.

In Section 3.9.1 an improvement to the routing protocol was briefly mentioned. The second improvement to the routing protocol concerns connections that have already been set up between switches in the form of logical links (links that have been used to set

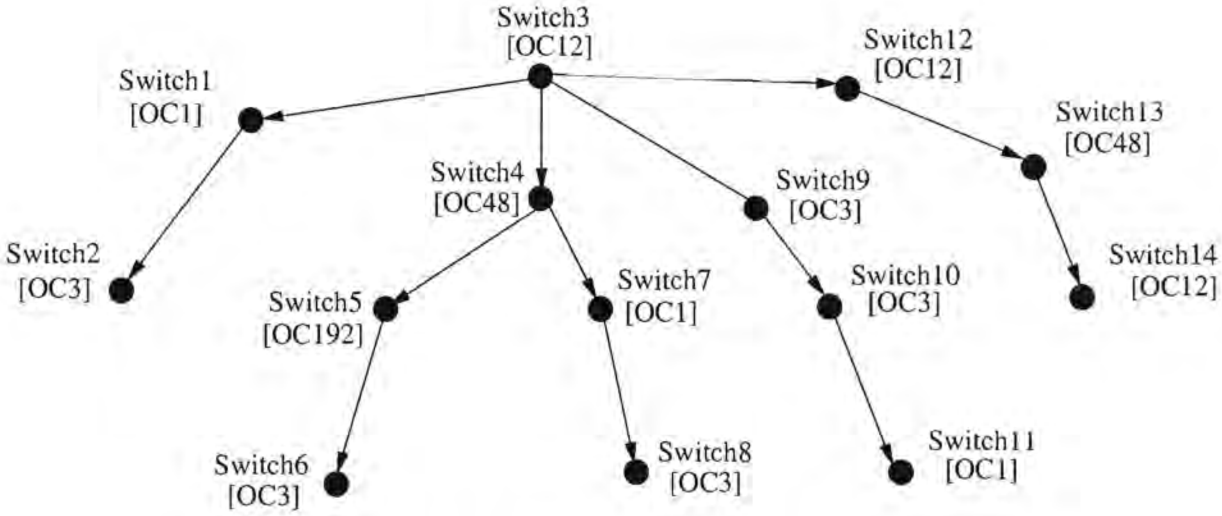


Figure 4.5: Shortest path tree constructed by Switch3

Destination node address	Data rate OC1	Data rate OC3	Data rate OC12	Data rate OC48	Data rate OC192
	Next hop option 1	Next hop option 1	Next hop option 1	Next hop option 1	Next hop option 1
Switch1	Switch1	Switch1	Switch1	Switch1	Switch1
Switch2	Switch1	Switch1	Switch1	Switch1	Switch1
Switch4				Switch4	Switch4
Switch5					Switch4
Switch6					Switch4
Switch7				Switch4	Switch4
Switch8	Switch7	Switch7	Switch7	Switch7	Switch7
Switch9		Switch9	Switch9	Switch9	Switch9
Switch10		Switch9	Switch9	Switch9	Switch9
Switch11		Switch9	Switch9	Switch9	Switch9
Switch12			Switch12	Switch12	Switch12
Switch13			Switch12	Switch12	Switch12
Switch14			Switch12	Switch12	Switch12

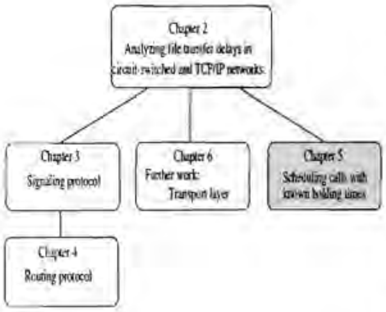
Table 4.9: Routing table for Switch3 (OC1 and OC3 rates cannot be used)

up lower rate connections over switches with high cross connect rates) and provisioned connections. An improvement to the routing protocol would be to exchange information regarding available bandwidth on connections that have already been set up, not just available bandwidth of the interfaces. For example, if a unidirectional OC12 connection is set up between two switches (from switch I to switch I+1), and only one OC1 connection is currently active on it, switch I can advertise that it is directly connected to switch I+1 (even if there are switches between them), and the cost of the link is again a function of the available bandwidth of the connection.

Chapter 5

Scheduling calls with known holding times

5.1 Introduction


Chapter 2
Analyzing file transfer delays in circuit-switched and TCP/IP networks
Chapter 3
Signaling protocol
Chapter 6
Further work:
Transport layer
Chapter 5
Scheduling calls with known holding times
Chapter 4
Routing protocol

This chapter explores the option of queueing calls in connection-oriented networks instead of blocking them when network resources are unavailable. A simple call queueing algorithm would be to hold up call setup messages at each switch along an end-to-end path until resources become available. This simple call queueing algorithm was modeled in Chapter 2, Section 2.2.1. This scheme will be shown to suffer from poor network utilization and long call queueing delays. However, if calls have known holding times, it is possible to design call scheduling algorithms that result in reduced call queueing delays and improved network utilization. We propose algorithms for such call scheduling and demonstrate the quantitative benefits of algorithms that exploit knowledge of call holding times.

In a simple connection setup scheme, a call setup message is held up at each switch sequentially along the end-to-end path until resources become available at each switch. There are two problems with this simple approach. *Firstly*, the total queueing delay incurred waiting for network resources can become rather large due to the sequential waiting period at each switch on the end-to-end path. *Secondly*, while call queueing schemes in general improve bandwidth utilization compared to call blocking schemes, the

kT_{wait} scheme will not achieve the maximum improvement possible. The kT_{wait} scheme was first used in Miyahara et al's paper [7] for the comparison between circuit-switching and CL packet-switching for large file transfers. In this scheme the signalling message requesting the connection is placed in a queue at each switch to wait for the required resources to be released. Only when the connection request reaches the head of the queue, and its requested resources are available, will the request be passed on to the next switch. Hence the term kT_{wait}. The reason an improvement in bandwidth utilization can be expected in call queueing schemes is that by having buffers to queue calls, less bandwidth is needed than for a strictly call blocking scheme. The reason why the kT_{wait} scheme does not take full advantage of this improvement is that upstream switches hold resources while waiting for downstream switches to admit a call instead of using the wait period to admit shorter calls that only traverse upstream segments.

To overcome these two problems, we started looking for ways to improve the call queueing algorithm. This led us to consider options where the switches agree upon a delayed start time for a given call c, and allow other calls sharing segments of the end-to-end path of call c to use the network resources for other calls before call c starts. This would decrease call queueing delays and allow for the utilization gains of call queueing to be realized. However, scheduling calls for a delayed start time with mutual agreement at all the switches on the end-to-end path is only possible if call holding times are known. A switch cannot guarantee that resources will be available for a new call c at some later point in time if it does not know when existing calls will complete. Hence, we focus on the call queueing/scheduling problem for calls with known holding times that can be identified when satisfying the following two characteristics:

- The data from the sending end of the call should be "stored," as opposed to "live."
- The CO network should use preventive congestion control as opposed to reactive.

We continued to design a call scheduling algorithm when call holding times are known. Such an algorithm could coexist with a traditional call admission control algorithm for calls with unknown holding times by partitioning network resources for calls with known holding times from resources for calls with unknown holding times.

This proposed method for scheduling connections can be used in any type of connection-oriented (CO) network. CO networks are networks where resources are reserved for a connection between end hosts before any data transmission may start. These networks can either be packet-switched (for example ATM or X.25 networks) or circuit-switched (for example SONET or WDM networks).

The motivating applications for such an algorithm are described in in Section 5.2. Section 5.3 describes the proposed algorithm as applied to a simple one-switch network, Section 5.4 continues by explaining some extensions to the algorithm, which includes the algorithm's behaviour in a network where multiple switches are involved. Some solutions to the distributed computing problem are proposed in Section 5.5 and Section 5.6 provides the results of simulations run to determine the feasibility of the solutions. Section 5.7 introduces some problems that still need to be solved. Conclusions are made in Section 5.8.

5.2 Motivating applications

While the motivation for this work came from our interest in supporting end-to-end large file transfers on high-speed circuit-switched networks, we considered the question of whether there are other applications for which call holding times are known. Two characteristics of calls have been identified that allows us to determine the call holding time:

- The data from the sending end of the call should be "stored," as opposed to "live."
- The CO network should use preventive congestion control as opposed to reactive.

Consider the first characteristic. Table 5.1 shows that the sending end and consuming end of any two-party (as opposed to multi-party) data transfer can each be "live" or "stored." If both ends are live and the communication is bidirectional, we classify such sessions as *interactive*. An example of this category is telephony, where the call holding time is unknown. Given that both ends are "live" and both ends can send traffic, the call could hold for any length of time. If both ends are live, but the communication is unidirectional, we refer to this case as a *live streaming* data transfer. An example is listening to/viewing a live radio/TV broadcast. The next category shown in Table 5.1 is *recording*, where the source is live, but the receiver is a storing device. In both the *live streaming* and *recording* categories, the call holding time may or may not be known since it depends on the duration of the live "event." For e.g., the duration of coverage of a live parade could be known a priori and advertised by the audio/video distributor; on the other hand, the duration of a sporting event, for e.g., a tennis match, is not known beforehand. Therefore, in general we assume that if the sending end is "live," call holding times are unknown.

Sending end \ Consuming end	Live	Stored
Live	Interactive/Live streaming	Recording
Stored	Stored streaming	File transfers

Table 5.1: Classification of data transfers

However, if the sending end is "stored," call holding times are known if the CO network is used with preventive congestion control (rather than reactive). For example, transferring a stored file from a source to a receiver where it is also stored for later consumption (classified as *file transfers* in Table 5.1) on a TCP/IP network results in an unknown holding time since this network uses reactive congestion control. On the other hand, if the same file is transferred on a circuit established through a circuit-switched network, the holding time can be determined from the file size, the data rate of the circuit, and propagation delays. Similarly, applications in which stored audio or video clips (e.g., in a web-based class lecture or Video on Demand (VOD)) are consumed live (e.g., by a student listening) are classified as *stored streaming* in Table 5.1. If such data is sent on a packet-switched CO network (to take advantage of silences, packet-switched networks are better for this class of applications), and the network supports preventive congestion control, then the call holding time can be estimated with a high degree of confidence. For example, if an ATM Variable Bit Rate (VBR) connection is used rather than an Available Bit Rate (ABR) connection, the call holding time can be accurately predicted given the exact traffic characterization of the stored audio or video file.

Thus, there are a number of applications, which when used on certain types of networks, have deterministic call holding times. Hence called known holding times.

Example applications from the categories presented in Table 5.1 that may benefit from this scheduled mode of connection setup are as follows:

- A frequently used application in networks is **file transfers**, which is an example of an application that transfers data from a stored source to a destination where the data is also stored. A large file transfer has no intrinsic burstiness associated with it, and is hence best handled by a circuit switched network [5], [6] and [7]. This is relative to connectionless packet switching where the packet header overheads, acknowledgements, congestion control mechanisms, buffering etc. have an impact on the end-to-end transfer time. Knowing the file size f and the data rate *rate* of the connection, the holding time h of a connection that will be used to transfer the file(s) is given by $h = \frac{f+overhead \times f}{rate} + T_{prop}$ where *overhead* is the overhead added by the various protocol layers and T_{prop} is the propagation delay from the sending host to the receiving host. Examples of large file transfers are application downloads, downloads from web sites with significant multimedia content (even with compression, video, audio and image files tend to be large), downloads of books, technical specification documents, etc. Large file transfers also occur in the mirroring of web sites. For example, Metalab's Linux archive is reported to be mirrored by more than 70 sites across the world [23].

- A second example of an application in which calls have known holding times is **video on demand** for stored video files. If the video file is sent out from a stored source and consumed live, this is an example of a *stored streaming* application. The implication of live consumption at the receiving end is that it is more efficient to use a variable rate connection for the video transfer given that most compression techniques take advantage of changing scene conditions. For example, during video transfer it is possible that there is a time frame during which there is a still image with only audio. Using a compression algorithm, this time frame would require less capacity than a normal time frame. Now, together with the holding time, traffic patterns can be associated with time frames allowing networks accommodating variable rate connections, for example ATM networks, to manage its resources more efficiently by setting up a connection that may vary in capacity and thus suit the transfer more efficiently. That is, the CO packet-switched network that implements preventative congestion control mechanisms reserves resources for the connection that vary over time. It is now possible for the network to decide to reduce the capacity allocated to one connection during a time frame and allocate the resources to another connection only during that time frame. Thus, even variable bit rate connections could have known holding times. A scheduling algorithm can, in principle, be designed for such packet-switched networks to take advantage of this knowledge and schedule calls for later start times instead of blocking calls when it does not have the resources available at the time when the request arrives.

- Leased lines are used commonly to interconnect two routers. This means that all traffic between the two routers concerned traverse a provisioned connection (also known as leased line) set up between them. Currently, the resources allocated to such a provisioned connection (for example, bandwidth) is for the worst case, i.e. the highest expected traffic. For example, consider two routers connected through a SONET network. Assume that for the most of the time an OC12 connection is sufficient to carry the inter router traffic. But, from 11am to 3pm the traffic is predicted to increase significantly requiring an OC48 connection. To be able to handle this worst case, the provisioned connection needs to be an OC48. This has the consequence that a lot of bandwidth is wasted during the time when just an OC12 would be sufficient. Making use of scheduled connection setup, it is possible to use traffic characteristics to schedule the setup of one or more extra connections between the routers during peak usage. Continuing our example, it should be possible to provision an OC12 connection for "permanent" usage, and schedule the setup of three additional OC12 connections from 11am to 3pm every day. Knowing the duration of the highest traffic load introduces a third example application, which is **leased lines with known holding times**.

5.3 Proposed algorithm

The general idea behind this method of admitting connections with known holding times is to have switches maintain a time-varying available capacity on each of its interfaces that reflects the scheduled start times of all admitted connections. Making use of this information the connection admission control (CAC) module of each switch, which is responsible for the management of the resources available to connections passing through the switch, determines the earliest possible time when a newly arriving connection with a known holding time can become "active".

An "active" connection is a connection that is currently transferring data, while an "admitted" connection is one that is either active or scheduled to become active at some point in time. As the simplest case, we explain our CAC algorithm using a one switch network as shown in Figure 5.1. Calls are generated by an end host connected to the switch to another end host on the same switch with a given desired capacity (c) and a known holding time (h). The CAC module in the switch, as shown in Figure 5.2, keeps a time-variant available capacity $a_i(t)$ function for each outgoing interface i. Based on the destination of the connection a set of candidate outgoing interfaces (I) is selected from a routing table maintained by the routing module (see Figure 5.2). Next, based on the

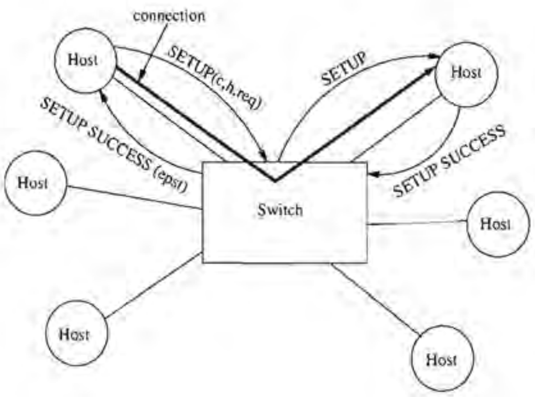


Figure 5.1: A simple one switch network

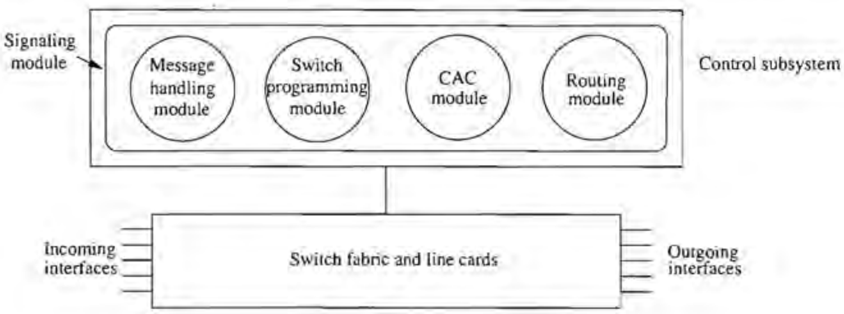


Figure 5.2: Block diagram of a switch

desired capacity (c) and holding time (h) for the connection, the earliest possible start time ($epst$) and interface (i) is determined by the CAC module using (5.1).

For each $j \in I$ determine

$$epst_j \geq \text{ current time} \wedge a_j(t) \geq c, epst_j \leq t \leq epst_j + h$$

$$epst \triangleq epst_i \text{ where } epst_i = min\{epst_j \mid j \in I\} \quad (5.1)$$

$$a_i(t) \leftarrow a_i(t) - c, \text{ for } epst \leq t \leq epst + h \quad (5.2)$$

A switch is now able to respond with an earliest possible start time at which the end host that requested the connection may start transmission. The time-varying available capacity function for the chosen interface i is updated using (5.2) to reflect the newly scheduled connection. Once the connection's scheduled time arrives the switch programming module shown in Figure 5.2 programs the switch fabric, thereby activating the connection.

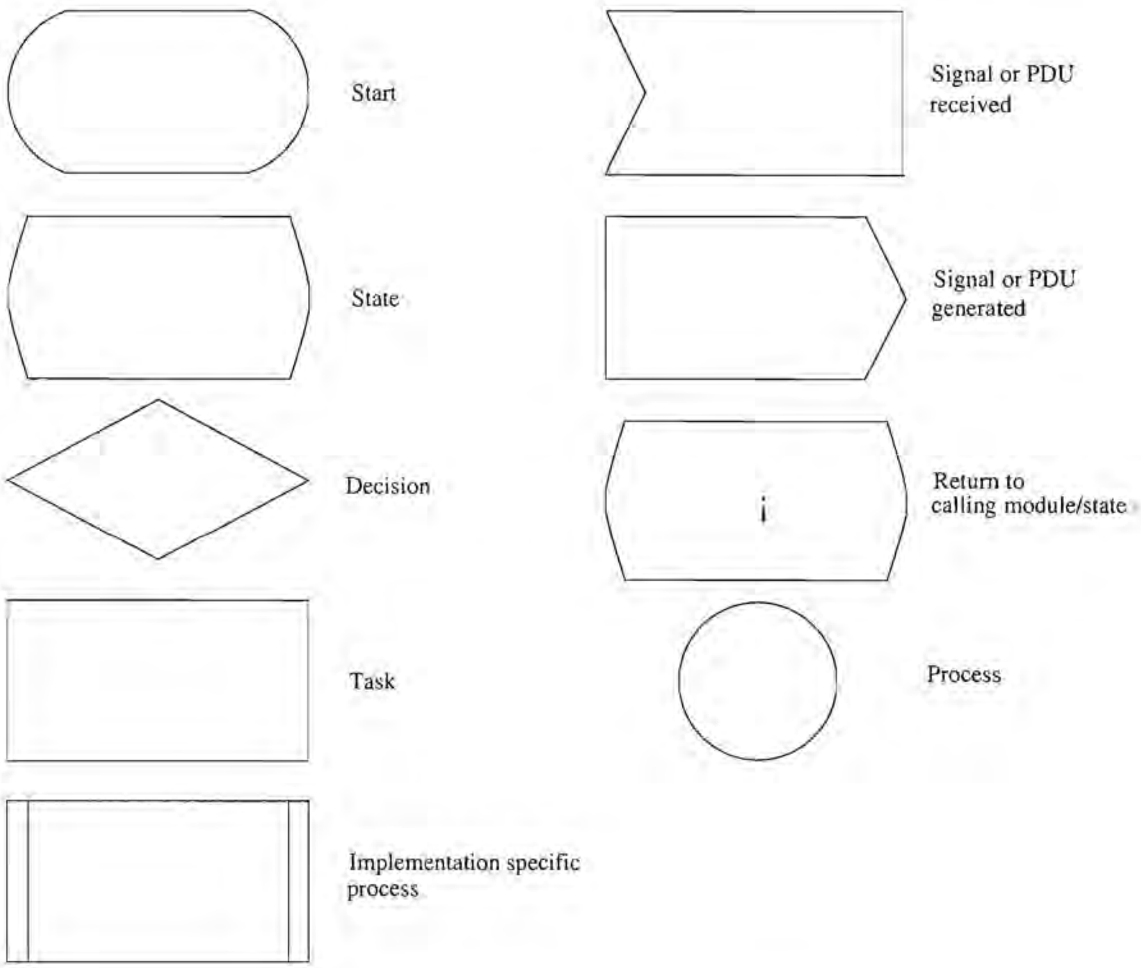


Figure 5.3: Keys used in flow diagrams

Using the keys presented in Figure 5.3, the message handling module (called from the signalling module) can be represented with the flow chart of Figure 5.4. This flow chart assumes an error-free operation. Figure 5.5 is a flow chart that indicates possible adjustments to be able to handle certain error conditions. Figures 5.6 to 5.9 depict the flow charts for processes included in Figure 5.4 and 5.5 assuming error conditions can occur.

In all these explanations, the destination host is treated as a "black box" which only returns "success" or "failure" when it replies to a connection request. That is, the switch includes the values of c,h and *epst* in the connection request sent to the destination, but the destination host does not change the value of *epst*. If the destination host can accept the connection its reply will indicate success. If it cannot accept the connection its reply shall indicate a failure. An example of the message exchanges for connection setup in a one switch network is given in Figure 5.1.

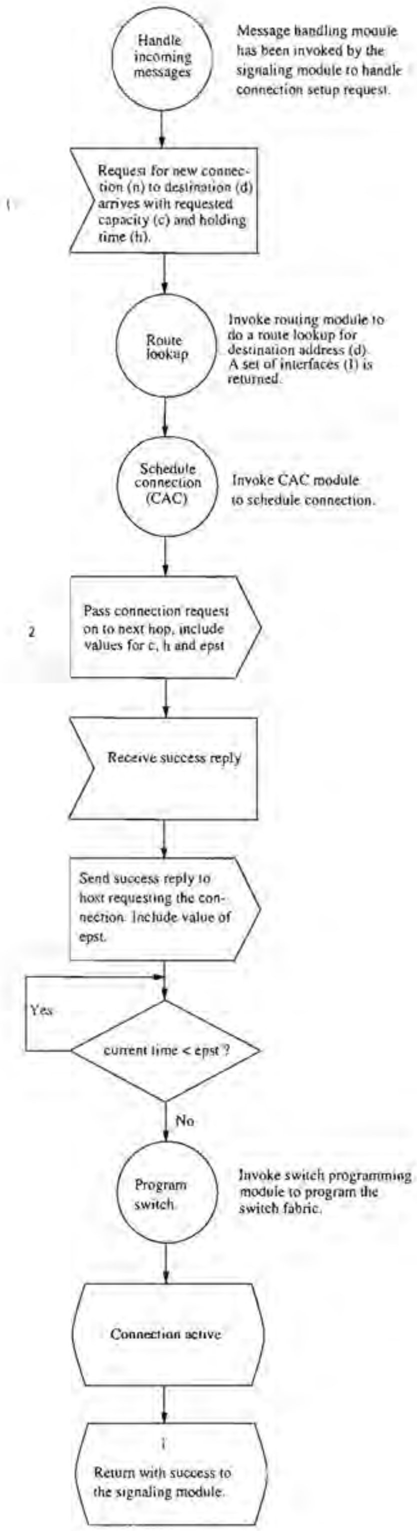

Handle incoming messages
Message handling module has been invoked by the signaling module to handle connection setup request.
1
Request for new connection (n) to destination (d) arrives with requested capacity (c) and holding time (h).
Route lookup
Invoke routing module to do a route lookup for destination address (d). A set of interfaces (I) is returned.
Schedule connection (CAC)
Invoke CAC module to schedule connection.
2
Pass connection request on to next hop, include values for c, h and epst
Receive success reply
Send success reply to host requesting the connection. Include value of epst.
Yes
current time < epst ?
No
Program switch
Invoke switch programming module to program the switch fabric.
Connection active
Return with success to the signaling module.

Figure 5.4: Flow chart for the message handling module in a one switch network

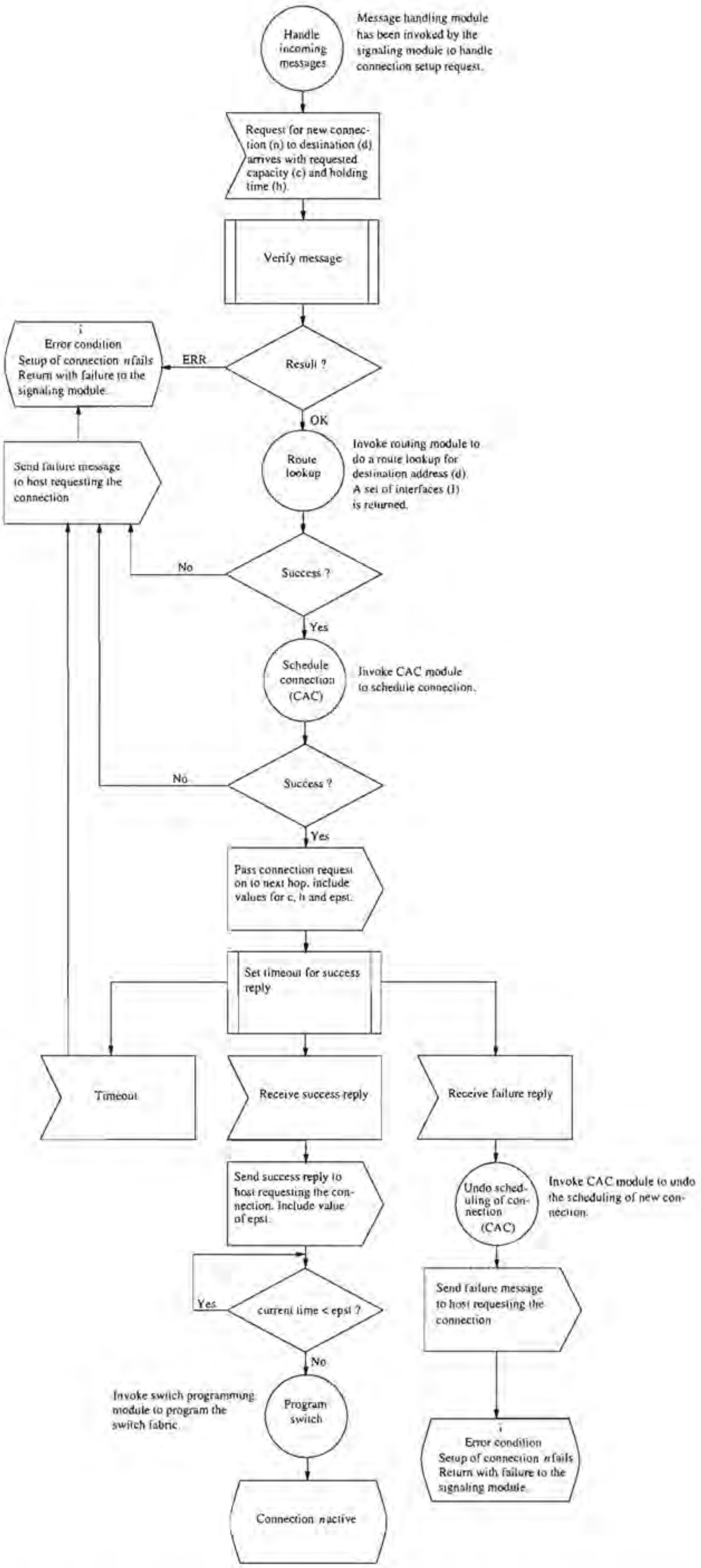

Handle incoming messages
Message handling module has been invoked by the signaling module to handle connection setup request.
Request for new connection (n) to destination (d) arrives with requested capacity (c) and holding time (h).
Verify message
Result ?
ERR
i
Error condition
Setup of connection n fails
Return with failure to the signaling module.
OK
Route lookup
Invoke routing module to do a route lookup for destination address (d). A set of interfaces (I) is returned.
Send failure message to host requesting the connection
Success ?
No
Yes
Schedule connection (CAC)
Invoke CAC module to schedule connection.
Success ?
No
Yes
Pass connection request on to next hop, include values for c, h and epst.
Set timeout for success reply
Timeout
Receive success reply
Receive failure reply
Send success reply to host requesting the connection. Include value of epst.
Undo scheduling of connection (CAC)
Invoke CAC module to undo the scheduling of new connection.
Yes
current time < epst ?
No
Send failure message to host requesting the connection
Invoke switch programming module to program the switch fabric.
Program switch
i
Error condition
Setup of connection n fails
Return with failure to the signaling module.
Connection n active

Figure 5.5: Flow chart for the message handling module in a one switch network, with error handling

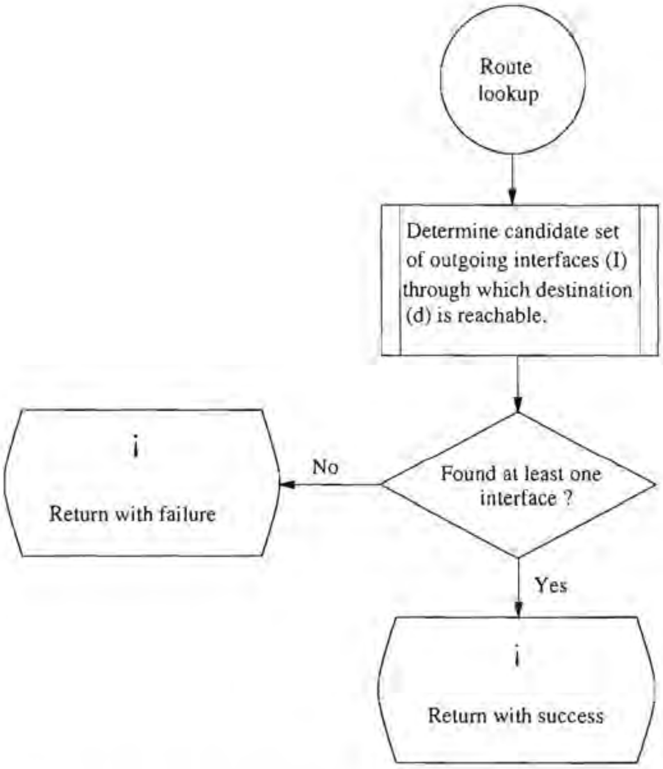

Route lookup
Determine candidate set of outgoing interfaces (I) through which destination (d) is reachable.
Found at least one interface ?
No
Yes
Return with failure
Return with success

Figure 5.6: Route lookup process included in the routing module

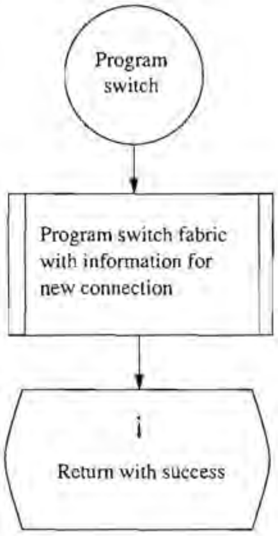


Figure 5.7: Process to program the switch fabric in the switch programming module

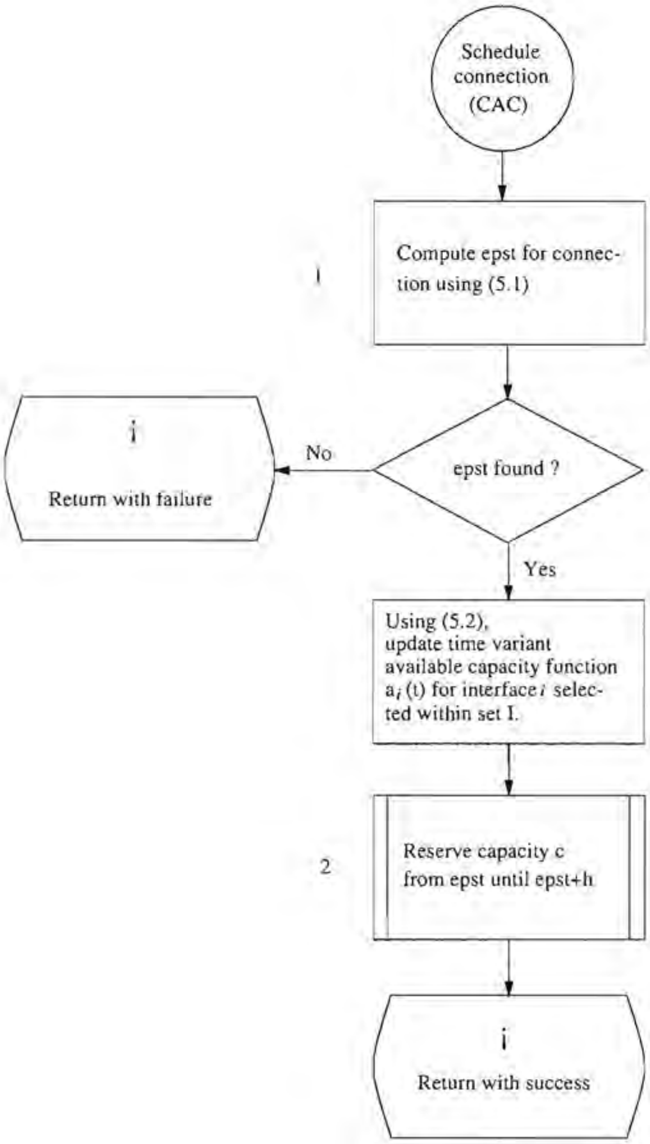

Schedule connection (CAC)
1
Compute epst for connection using (5.1)
epst found ?
No
Return with failure
Yes
Using (5.2), update time variant available capacity function a_i(t) for interface i selected within set I.
2
Reserve capacity c from epst until epst+h
Return with success

Figure 5.8: Scheduling process included in the CAC module for a one switch network

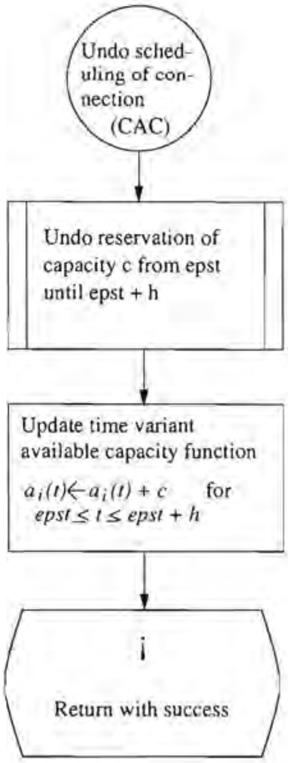

Undo scheduling of connection (CAC)
Undo reservation of capacity c from epst until epst + h
Update time variant available capacity function
$a_i(t) \leftarrow a_i(t) + c$ for $epst \leq t \leq epst + h$
i
Return with success

Figure 5.9: Process in the CAC module to undo the changes made by the scheduling process

5.4 Extensions

5.4.1 Request for start time included in connection request

Continuing with our example of a simple one switch network, an extension to the CAC algorithm is to allow the host requesting the connection to specify a desired start time, which could be different from the implied "immediate" start time in current signalling protocols. This is definitely needed in the third example application where the extra connections between the routers will only be needed during certain times. Requesting the start time could also be very useful in the other two example applications, for example only mirroring a web site from 10pm, or a customer requesting a movie to start at 8pm.

If a host is allowed to request a connection from a specific time *req*, *epst* is computed using (5.3).

For each $j \in I$ determine

$$epst_j \geq req \wedge a_j(t) \geq c, epst_j \leq t \leq epst_j + h$$

$$epst \triangleq epst_i \text{ where } epst_i = min\{epst_j \mid j \in I\} \tag{5.3}$$

The flow charts for this extension to the CAC algorithm can be depicted by changing block 1 in Figure 5.4 to "Request for new connection n to destination d arrives with requested starting time *req*, capacity c and holding time h" and replacing the flow chart for the scheduling process with Figure 5.10.

5.4.2 More than one switch in the network

Connections typically pass through multiple switches. A problem that arises in this case is that the switches may not compute the same value for *epst* due to the differences in their current state.

To explain the problem and its solution, we use the example shown in Figure 5.11 where Host1 requests a connection to Host2. The connection is shown to traverse through three switches. Assume the first switch, SwitchA schedules the connection and reserves resources for some time $[epst_A, epst_A + h]$, where h is the holding time and $epst_A$ is the earliest possible start time that the connection can be scheduled for at SwitchA. SwitchA needs to pass the appropriate information to the next hop switch, SwitchB, to prevent the latter from scheduling the connection for any $t < epst_A$. Now, assume SwitchB finds

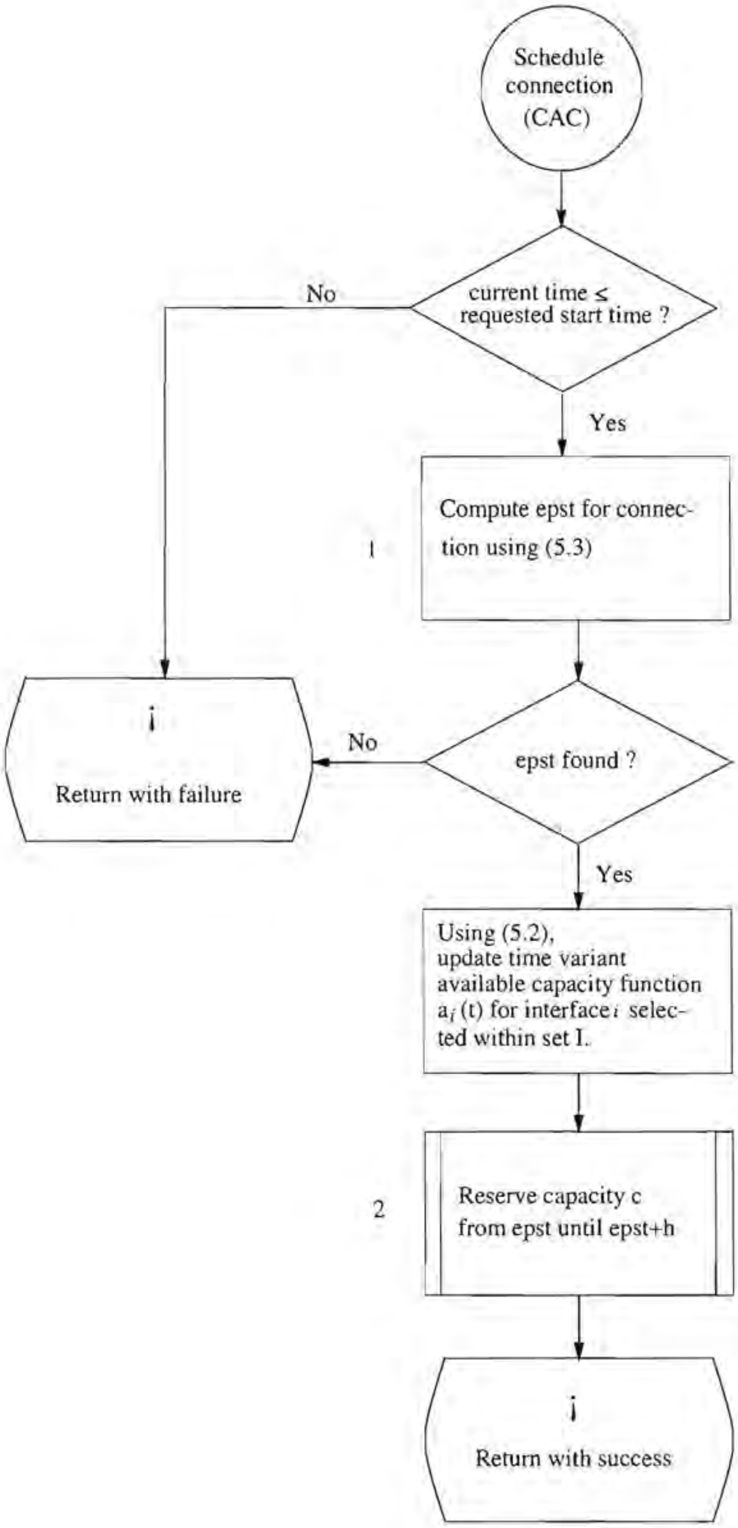

Schedule connection (CAC)
current time ≤ requested start time ?
No
Yes
Compute epst for connection using (5.3)
1
epst found ?
No
Yes
Return with failure
Using (5.2), update time variant available capacity function a_i(t) for interface i selected within set I.
2
Reserve capacity c from epst until epst+h
Return with success

Figure 5.10: Scheduling process (allowing for a host to request a start time) included in the CAC module for a one switch network

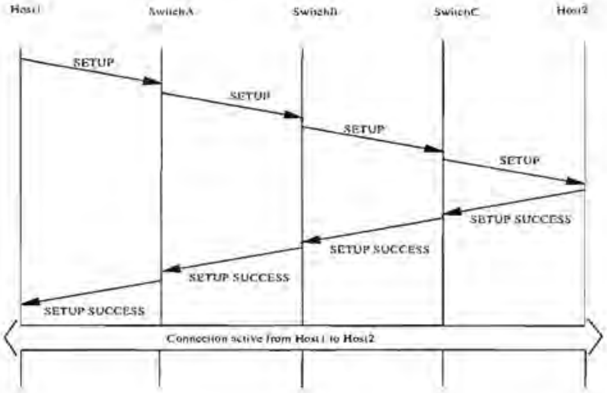


Figure 5.11: Successful setup of a connection

that it cannot schedule the connection for the same *epst* as SwitchA. Instead its *epst* is $epst_B$. This could cause a problem because SwitchA may not have resources available for this connection in the time interval $[epst_B, epst_B + h]$. This later starting time could have been allocated to some other connection at SwitchA. This is a distributed computing problem caused by needing cooperative actions at multiple switches for which we propose solutions in the next section.

5.5 Connection setup schemes

Currently, on-demand (referred to as "switched" mode of operation as opposed to "provisioned" mode) connections are set up in connection-oriented networks by using a *blocked* mode of operation. When a connection request arrives at a switch that does not have enough resources available, the request will not be *queued* until the resources become available. Instead the call is blocked and connection release procedures are initiated. The objective is to decrease the blocking probability of connections by making use of the known holding time to schedule a connection, thus providing the source that requested the connection with a later start time if resources are not immediately available.

This section compares four queued connection setup schemes. Two schemes, kT_{wait} and $kT_{wait} - T_{max}$, explore a queued mode of operation where the call holding time is not taken into account and connection setup requests are queued instead of blocked. Of all the schemes kT_{wait} is the only one that guarantees a zero blocking probability. The $kT_{wait} - T_{max}$ scheme extends the kT_{wait} scheme by allowing the source to include a latest acceptable start time for the connection in its connection request.

In the F scheme, the ingress switch selects an earliest possible start time (*epst*) for the connection such that is has resources available for the connection from *epst* to $epst+h+F$, where h is the holding time of the call and F is some large time value. It holds the resources for this connection for this large time period hoping that the downstream switches will select an *epst* within this large time period. If a large enough value of F is selected, the probability of success will be high.

In the *timeslots* scheme, the ingress switch selects multiple time ranges during which it can accommodate the call. Each downstream switch checks to see if it has available resources for one or more of these time ranges. Each switch along the path may narrow the time ranges until finally there exists at least one common time range (of length equal to the call holding time) during which the required resources are available at all switches.

An analogy can be drawn between these scheduling schemes and the scheduling of a meeting involving several people. With the meeting duration known to all involved, one way to schedule its start is for the first person to send out a memo stating that he/she has "the whole of next week" available. Another solution is for the memo to read, "I am free from 2pm to 4pm Monday, 8am to 1pm Tuesday, and the whole of the following week." Tentatively, the sender of the memo holds these time ranges for this meeting until a response arrives from all other participants. The first solution corresponds to the F scheme and the second solution corresponds to the *timeslots* scheme.

The F and *timeslots* schemes are call queueing schemes in that calls wait for resources, but they also block calls if the selected F value or number of time ranges is not large enough for success. These can be augmented with negotiations (i.e., multiple signalling exchanges on the same path), and/or searches on multiple paths to reduce call blocking probability.

5.5.1 The kT_{wait} scheme

The kT_{wait} scheme is the simplest queued connection setup scheme. In this scheme, switches do not take call holding times into account when connections are admitted. Instead, on receipt of a connection request it is placed in the queue of the interface that should be used by the connection. When the bandwidth requested by the connection request at the head of the queue becomes available, it is reserved for the connection and the connection request is passed on to the succeeding switch, or destination host. When the connection request reaches the destination host, a connection has already been set up from the source to the destination. If the destination host accepts the connection, it sends a success message back to the source. As this success message traverses the network towards the source the connection is usually marked as "established". On receipt of the success message, the source can immediately start with data transmission. If T_{wait} is the average time a connection request waits in a queue for resources to become available, and k indicates the number of switches on the connection, then the term kT_{wait} forms a large part of the total connection setup time, hence the name kT_{wait} as described in Chapter 2, Section 2.2.1.

5.5.2 The $kT_{wait} - T_{max}$ scheme

The second queued connection setup scheme is designed by slightly modifying the kT_{wait} scheme. When sending a connection request to its ingress switch, a source host might expect the connection to be established before a certain time. For example, the source host might request that a connection be established at most 50 seconds from when it requested it. This connection setup scheme also does not take the holding time into account when a connection is set up, but provides the source requesting the connection some control over the start time of the connection, hence named the $kT_{wait} - T_{max}$ connection setup scheme. The $kT_{wait} - T_{max}$ scheme works in the same manner as the kT_{wait} scheme except that the source host is allowed to include a time, T_{max}, in its connection request. The value of T_{max} indicates the latest possible time that the source would accept as a start time for this connection. As in the kT_{wait} scheme, switches place a connection request in the queue of the interface that should be used by the connection. The action taken when the connection request is at the head of the queue, and its requested bandwidth is available, differs in that the switch first checks if its current time is later than the time T_{max} included in the connection request. If the current time is later, connection release procedures are initiated.

5.5.3 The F scheme

The first queued connection setup scheme that takes advantage of the call holding time is called the F scheme. In this solution the requested resources are reserved for a longer period of time than its holding time, until reverse messages are received, during the connection setup procedure. In the forward phase of connection setup resources are held for a long enough time period that if a downstream switch selects a later value for *epst*, there is some likelihood that the earlier (upstream) switches of the path will have the resources available. Resources are held for this large time period only for the short duration it takes for the setup request to reach the destination and the reply (setup success) to return.

Using a large finish time during the scheduling of a connection works as follows: a host requests a connection from its ingress switch to a destination with the optional inclusion of a start time; the ingress switch selects a large time range (starting with the earliest time greater or equal to the requested start time) during which it is able to accommodate the connection. On receipt of a time range a switch searches for the largest time range inside the range it received from the preceding switch during which it can accommodate the connection. This time range is included in the connection request passed to the succeeding switch, or destination host.

The computation of *epst* changes in that a pre-assigned value F is added to the holding time by the ingress switch, where F is a very large extra holding time. When a host requests a connection to destination d from time *req* with capacity c and holding time h, (5.4) can now be used at the first (ingress) switch to compute *epst*. The time-varying available capacity function for the chosen interface is updated using (5.5) in which $T_{end} = epst + h + F$.

For each $j \in I$ determine

$$epst_j \geq req \wedge a_j(t) \geq c, epst_j \leq t \leq epst_j + h + F$$

$$epst \triangleq epst_i \text{ where } epst_i = min\{epst_j \mid j \in I\} \quad (5.4)$$

$$a_i(t) \leftarrow a_i(t) - c, \text{ for } epst \leq t \leq T_{end} \quad (5.5)$$

When the resources are reserved for this connection at the ingress switch, it indicates that this connection reserves capacity c from *epst* to $epst+h+F$. To enable all the switches in the connection to agree to the same value for *epst*, the value of *epst*, the holding time (h) and the time T_{end} are included in the connection request message passed to the second switch in the connection. The switch receiving this message shall handle this request as if a request for a connection from time *epst*, as included in the connection request, has been received.

The *setup request traverses* the network and each switch along the path computes a (possibly) new value for *epst* and looks for the largest time period $\leq F$ but $\geq h$, during which it will have enough resources available. If found, the requested resources are reserved for this connection (using (5.5)) for the period $(epst_{new}, epst_{new} + h + F_{new})$, and the connection request containing $epst_{new}$, $F_{new} + h$ and h are passed on to the next hop. If a switch is unable to find a time period during which it can accommodate the connection, the call is blocked and release procedures are initiated. If the destination host accepts the request for a connection from the source, the final value of *epst* is passed from switch to switch until the source is notified of the new value of *epst*.

As those *success reply messages* traverse the network to the source, all the switches change their resource reservation tables to reflect the new start time (the new value of *epst*), and the finish time to be $epst + h$. If $epst_{local}$ indicates the earliest possible start time as computed by a switch in the forward direction of the signalling procedure, and *epst* is the earliest possible start time it received in the success reply message, the time-varying available capacity function is updated using (5.6) and (5.7).

$$a_i(t) \leftarrow a_i(t) + c, \text{ for } epst_{local} \leq t < epst \tag{5.6}$$

$$a_i(t) \leftarrow a_i(t) + c, \text{ for } epst + h \leq t < T_{end} \tag{5.7}$$

The flow chart for the message handling module for the ingress switch is given in Figure 5.12. Block 1 in the scheduling process (given in Figure 5.10) changes to "Compute epst for connection n using Equation (5.4)", and block 2 changes to "Reserve capacity c from epst until epst + h + F.". The flow chart for the process "Update resource reservation and a(t)" is depicted in Figure 5.13.

Figure 5.14 shows an example of scheduled connection setup using the F scheme. For this example, the value of F is 5 hours. End host A requests a connection with a holding time of one hour starting at 6pm, switch 1 computes that it will have enough resources available from 6pm until 12pm, which it reserves. The setup request message traverses the network, and every time *epst* is increased due to the unavailability of resources. At switch 2 resources are reserved from 7pm until 12pm, and *epst* is set to 7pm. When the setup request message reaches end host B, *epst* is already 9pm. End host B accepts the connection and the success reply from switch 4 contains the final value of *epst* (9pm). As this reply message traverses the network, each node forming part of the connection now changes its respective resource reservations to indicate that the connection will be active from 9pm until 10pm.

This scheme might have the disadvantage of underutilizing the channels between switches. To be able to guarantee a low blocking probability, the ingress switch has to

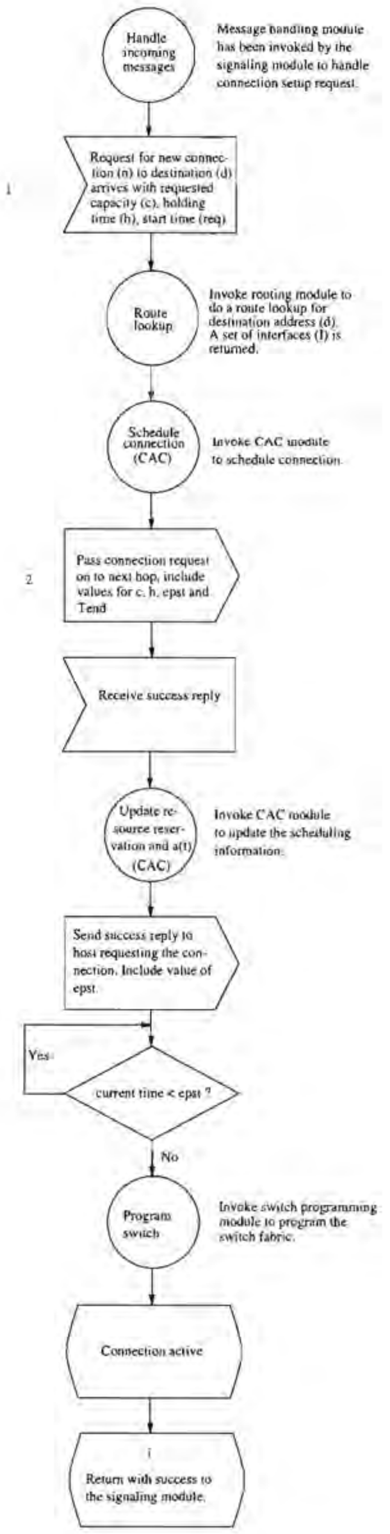

Handle incoming messages
Message handling module has been invoked by the signaling module to handle connection setup request.
1
Request for new connection (n) to destination (d) arrives with requested capacity (c), holding time (h), start time (req)
Route lookup
Invoke routing module to do a route lookup for destination address (d). A set of interfaces (I) is returned.
Schedule connection (CAC)
Invoke CAC module to schedule connection.
2
Pass connection request on to next hop, include values for c, h, epst and Tend
Receive success reply
Update resource reservation and a(t) (CAC)
Invoke CAC module to update the scheduling information.
Send success reply to host requesting the connection. Include value of epst.
Yes
current time < epst ?
No
Program switch
Invoke switch programming module to program the switch fabric.
Connection active
Return with success to the signaling module.

Figure 5.12: Connection setup at the ingress switch of a connection (F scheme)

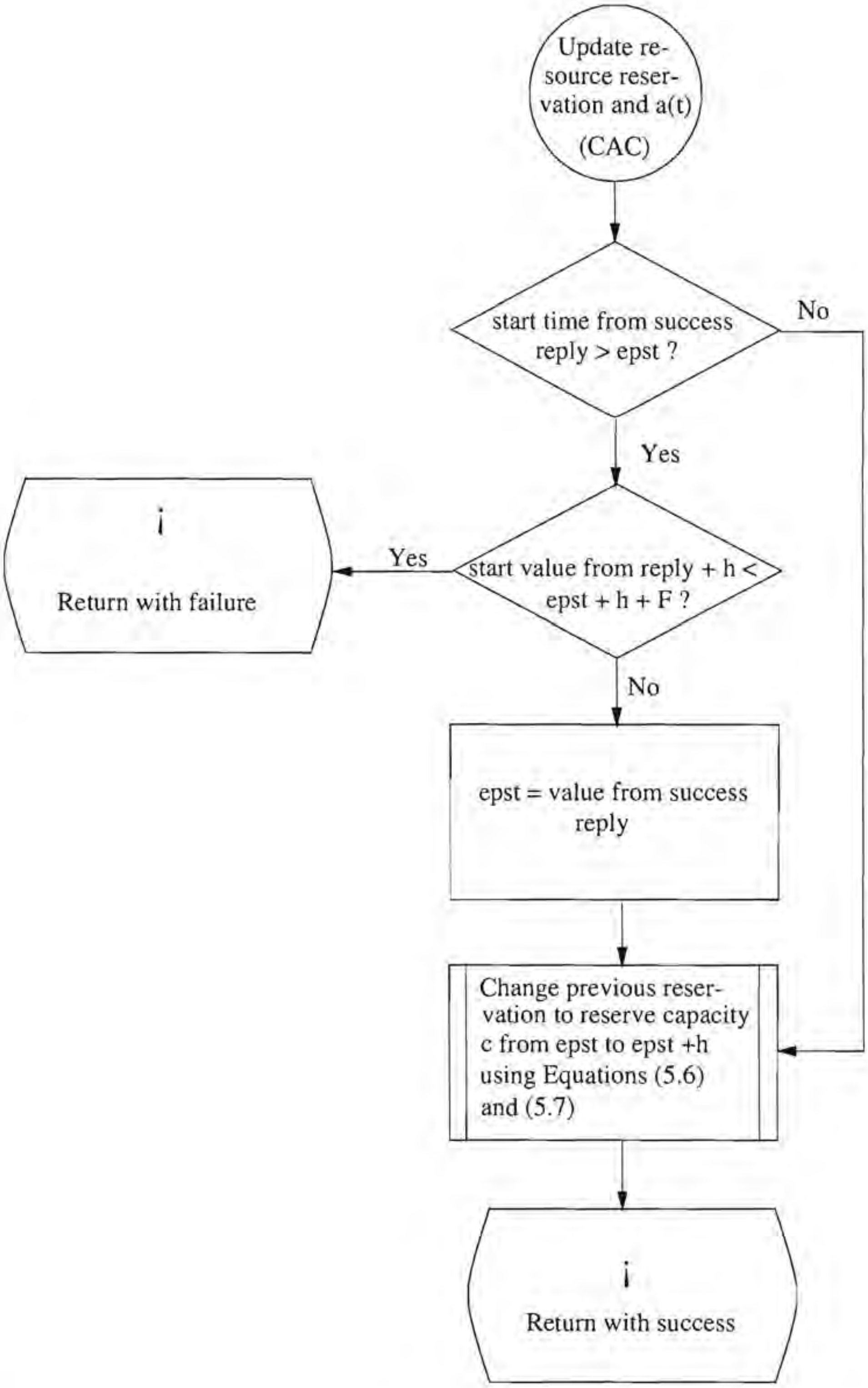

Update resource reservation and a(t) (CAC)
start time from success reply > epst ?
No
Yes
start value from reply + h < epst + h + F ?
Yes
Return with failure
No
epst = value from success reply
Change previous reservation to reserve capacity c from epst to epst +h using Equations (5.6) and (5.7)
Return with success

Figure 5.13: Process in the CAC module to update the changes made by the scheduling process in the reverse direction

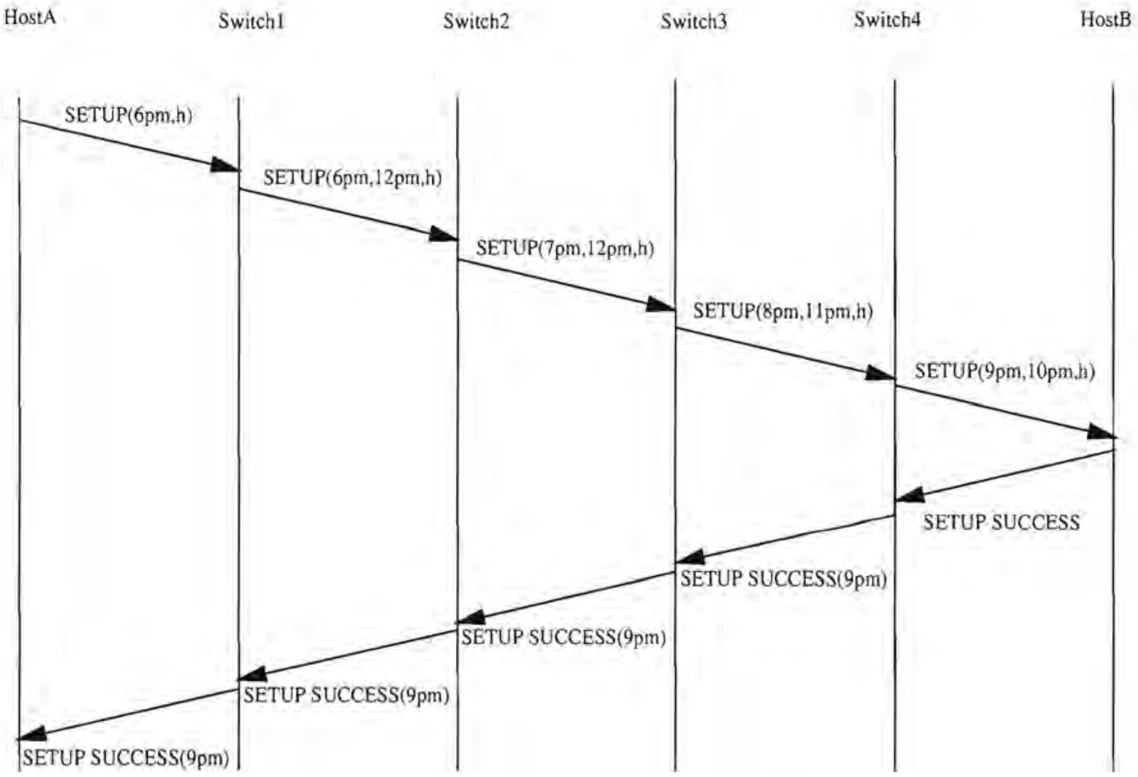


Figure 5.14: Example of scheduled connection setup

make use of a large value for F. Choosing a large value for F will increase the probability that downstream switches will find a time period corresponding to the one provided by the ingress switch during which it has enough resources available for the connection. The disadvantage of a large value for F is that the ingress switch might have resources available for the connection during time periods that are smaller than F, but large enough to handle the connection. These time periods will be ignored by this scheme, and will result in low utilization of the channels between switches. For example, Figure 5.15 describes the available bandwidth function for an OC1 interface if the current time is 2pm. Consider an F value of four hours. If the F scheme is used and a connection that has to be handled by this interface arrives, requesting an immediate start time and indicating a holding time of one hour; the earliest possible time this interface will be able to handle the connection will be at 10pm - even though resources are available from 3pm until 5pm. The next scheme, *timeslots*, shall attempt to solve this problem.

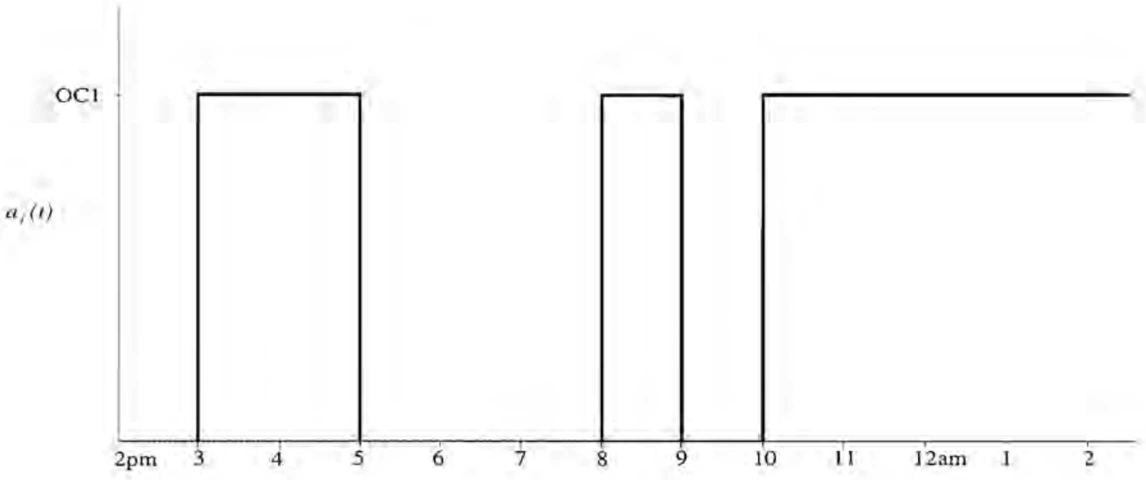


Figure 5.15: Available bandwidth of Switch1, interface i

5.5.4 The *timeslots* scheme

An improvement to the F scheme that will solve the aforementioned problem of utilization, without increasing the call blocking probability significantly, involves the use of several time ranges. Including more than one time range in the connection request will increase the expected utilization of the channels without increasing the call blocking probability significantly. In this scheme the ingress switch does not rely on a large value (F in the previous scheme) to admit a call, but rather a number n indicating the number of time ranges it should include in the connection request passed on to the succeeding switch.

A time range j is identified by a start time t_{js} that is equal to or larger than the requested time (req) and the end time t_{je} that is larger or equal to $t_{js} + h$, during which the switch has resources available to handle the connection. If I is the set of interfaces selected by the routing module as described in Section 5.3, the ingress switch selects n different time ranges by using (5.8).

Find n time ranges such that,

$$t_{1s} \geq req,$$

$$t_{1s} < t_{1e} < t_{2s} < t_{2e} < t_{3s} < t_{3e} < \ldots$$

$$<_{tns}< t_{ne},$$

$$t_{je} - t_{js} \geq h, \forall j, 1 \leq j \leq n, \text{and}$$

$$a_i(t) \geq c, t_{js} \leq t \leq t_{je}, \forall j, 1 \leq j \leq n$$

for any $i \in I$ (5.8)

Upon finding n time ranges, (t_{1s}, t_{1e}), (t_{2s}, t_{2e}),..., (t_{ns}, t_{ne}), the required resources are reserved using (5.9).

$$a_i(t) \leftarrow a_i(t) - c, \text{ for } t_{js} \leq t \leq t_{je}, \text{ for } 1 \leq j \leq n \quad (5.9)$$

The n time ranges are sent in the connection setup message to the next switch along with the holding time h. On receipt of a connection request containing time ranges, an intermediate switch attempts to admit the call during each of the time ranges or any part of each range greater than or equal to the holding time. The matching time ranges are passed on to the succeeding switch. If no matching time ranges are found, the call is blocked and connection release procedures are initiated.

For example, if Figure 5.15 describes the available bandwidth function of an OC1 interface of Switch1 and a connection request arrives requesting an immediate start time for an OC1 connection lasting one hour, the *timeslots* scheme enables the switch to pass the connection request to the succeeding switch containing the time ranges [3pm,5pm], [8pm,9pm] and [10pm,∞] in the case when $n = 3$. On receipt of a connection request containing time ranges, a switch shall attempt to admit the call during each of the time ranges or any part of it greater than or equal to the holding time. The matching time ranges (of which there could now be more than n) are passed on to the succeeding switch. If no matching time ranges are found, connection release procedures are initiated. For example, Figure 5.16 shows the available bandwidth function of Switch2 before it received the connection request from Switch1. On receipt of the connection request, Switch2 shall admit the call during the timeslots [4pm,5pm], [10pm,12am] and [1am,2am]. The new time ranges are then passed to the succeeding switch.

If, after passing through all switches on the end-to-end path a time range greater than the holding time is found, the destination is offered the call. If it accepts, it creates a success message with the start time of the earliest time range in the connection request it received (*epst*). As this message traverses the network each switch releases reserved

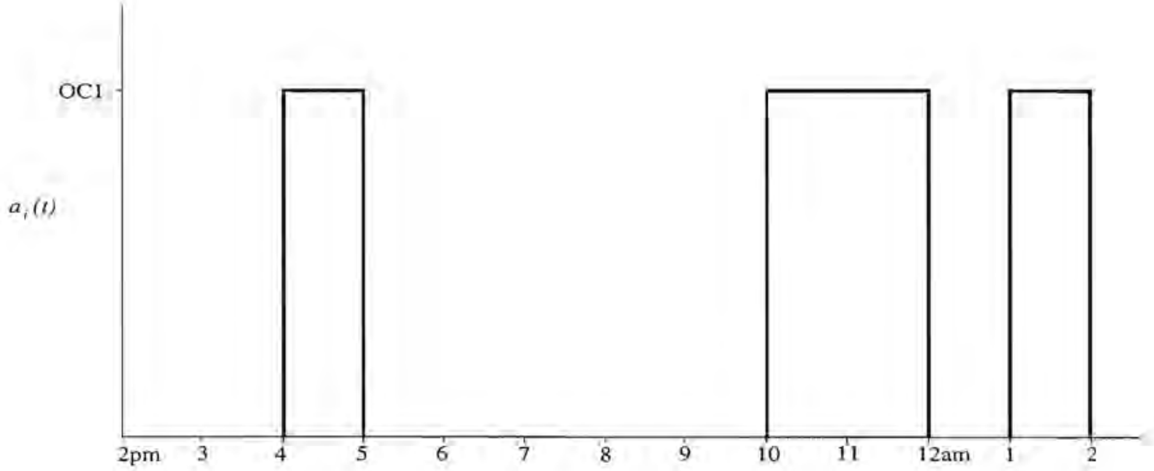


Figure 5.16: Available bandwidth of Switch2, interface i

resources during the unselected time ranges as shown in (5.10).

```
For $1 \le j \le n$,
    If $t_{js} \le epst \le t_{je}$ then
            $a_i(t) \leftarrow a_i(t) + c$ for $t_{js} \le t < epst$ and
            $a_i(t) \leftarrow a_i(t) + c$ for $epst + h < t \le t_{je}$
    else
        $a_i(t) \leftarrow a_i(t) + c$, for $t_{js} \le t \le t_{je}$    (5.10)
```

In the above description, we assumed that the number of time ranges sent from each switch is n, a constant. However, it is feasible to allow intermediate switches to create more time ranges than the number n of time ranges selected by the ingress switch. This could happen if resources are not available at an intermediate switch for the whole duration of a time range selected by the ingress switch, but instead parts of the time range have available resources at that intermediate switch. Also, the number of time ranges selected at an intermediate switch could be fewer than n if it does not have the resources available during some of the time ranges.

5.6 Simulation and results

5.6.1 Network model

A comparison of the four different connection setup schemes was done using simulation with the help of the discrete event simulator OPNET Modeler [24]. The network model studied is presented in Figure 5.17. It is a circuit-switched connection-oriented network implementing a very simplified CAC mechanism in which each channel is able to provide one "bandwidth unit", which is requested by a connection. An analogy can be drawn between this network and a SONET (Synchronous Optical NETwork) network in which all the channels are OC1 channels. The network model consists of four switches and all the links shown in Figure 5.17 are unidirectional channels providing one bandwidth unit. In general networks will have much higher channel capacities. We chose only one bandwidth unit for the interfaces to decrease the startup settling time of the simulations. Connections were set up and released between *Source* and *Dest*, and between *srcx* and *destx*. The connections between *Source* and *Dest* were studied (hence called *study traffic*), and the connections between *srcx* and *destx* were created as "interference traffic" to create a more realistic network model. Interarrival times of the connection setup requests and the holding times of the connections are assumed to be exponentially distributed. The destinations always accepted any connection, allowing us to concentrate on the CAC performed by the switches.

The mean call interarrival time and mean holding time for the study traffic was kept constant through all the simulations while the mean call interarrival time and holding times of the interference traffic was varied. This allowed us to simulate the behaviour of the different connection setup schemes' under different load conditions. The mean call interarrival time used by *Source* was 25 seconds and the mean holding time was 5 seconds. This computes to a mean load of 20% introduced to the network by *Source*.

Table 5.2 presents the combinations of mean holding time and mean interarrival time used for the interference traffic generated by *src1*, *src2* and *src3*. During each simulation run each of the sources generating interference traffic did so using the same mean holding time and mean interarrival time. With these different load conditions, the OC1 channels between switches will experience load varying from 25% to 95%, while the load on the channel between *Switch4* and *Dest* is kept constant at 20%.

Table 5.3 presents the parameters used for each scheduling scheme. There are no parameters for the *kTwait* scheme. The parameter D_{max} for the $kT_{wait} - T_{max}$ scheme

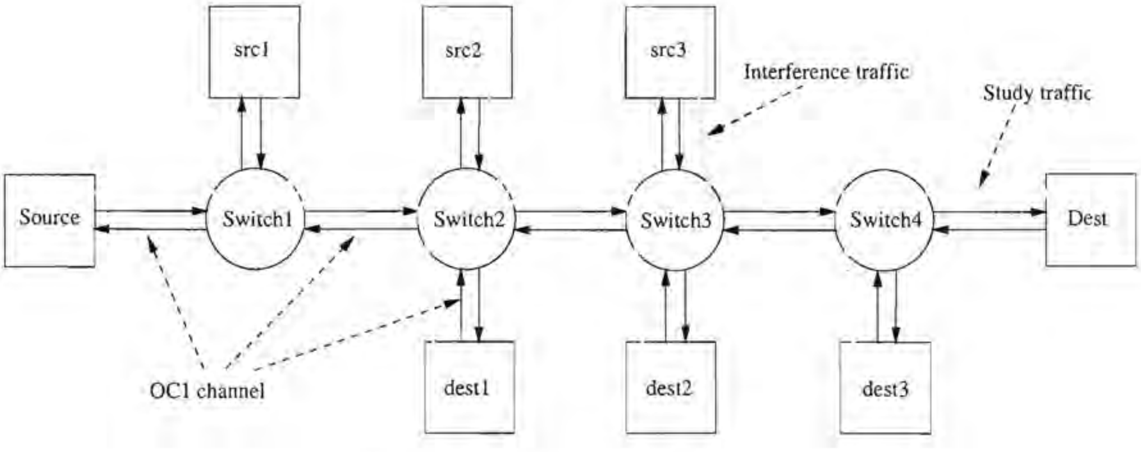


Figure 5.17: Network model

shown in Table 5.3 indicates the maximum queueing delay that is acceptable. In other words, by adding D_{max} to the time when the connection request is sent out by the source we get the value of T_{max}. Each parameter from Table 5.3 combined with each combination of mean interarrival time and mean holding time comprised the parameters for one simulation run. For example, one simulation consisted of the scheduling scheme being *timeslots*, the number of time slots being 4, with the mean interarrival time of the interference traffic of 20 seconds and the mean holding time of the interference traffic of 9 seconds. Each simulation run simulated one hour of network activity and was repeated 201 times, each time with a different seed for the random variables.

5.6.2 Results

The results of most interest are the *utilization* of the channels, the *call blocking probability*, and the *start time delay* returned by the network when a request for a connection is made. In the *timeslots* and F schemes, the channels are either free or in use by a connection. In the two kT_{wait} schemes the channels could be in one of three states: free, reserved or in use. The second state (reserved) is used to indicate that the channel has been reserved for a connection but the success reply has not yet been received, which means the channel is not "in use" (used to transfer data from the source to the destination). Hence, the utilization of the channels are measured by computing how much time a channel is "in use" by connections. The F and *timeslots* schemes also have a period in which resources are reserved but not in use, i.e., between sending the setup message and receiving the success reply. However, this time is far smaller (in the order of ms) than the wait time for

Mean interarrival time	Mean holding time	Load introduced by interference traffic
100	5	5%
50	5	10%
40	6	15%
25	5	20%
20	5	25%
20	6	30%
20	7	35%
20	8	40%
20	9	45%
10	5	50%
20	11	55%
20	12	60%
20	13	65%
10	7	70%
20	15	75%

Table 5.2: Different loads introduced by interference traffic

resources (in the order of sec), and is hence neglected.

The kT_{wait} scheme does not block any calls. For all the other schemes, the call blocking probability is estimated by computing the ratio between the number of connections requested by *Source* and the number of connections released by any intermediate switch.

The *start time delay* is the time difference between when *Source* requests a connection (*req*) and the value of *epst* as returned by the network. In the kT_{wait} and $kT_{wait} - T_{max}$ schemes, where there is no *epst* present, the *start time delay* is the difference between when the connection request is sent and the time when the network replies with success.

Parameter	Values
F (F scheme)	20, 50 and 100 seconds
t (*timeslots* scheme)	2, 3 and 4
D_{max} ($kT_{wait} - T_{max}$ scheme)	100, 150 and 200 seconds

Table 5.3: Parameters used in simulation of connection setup schemes

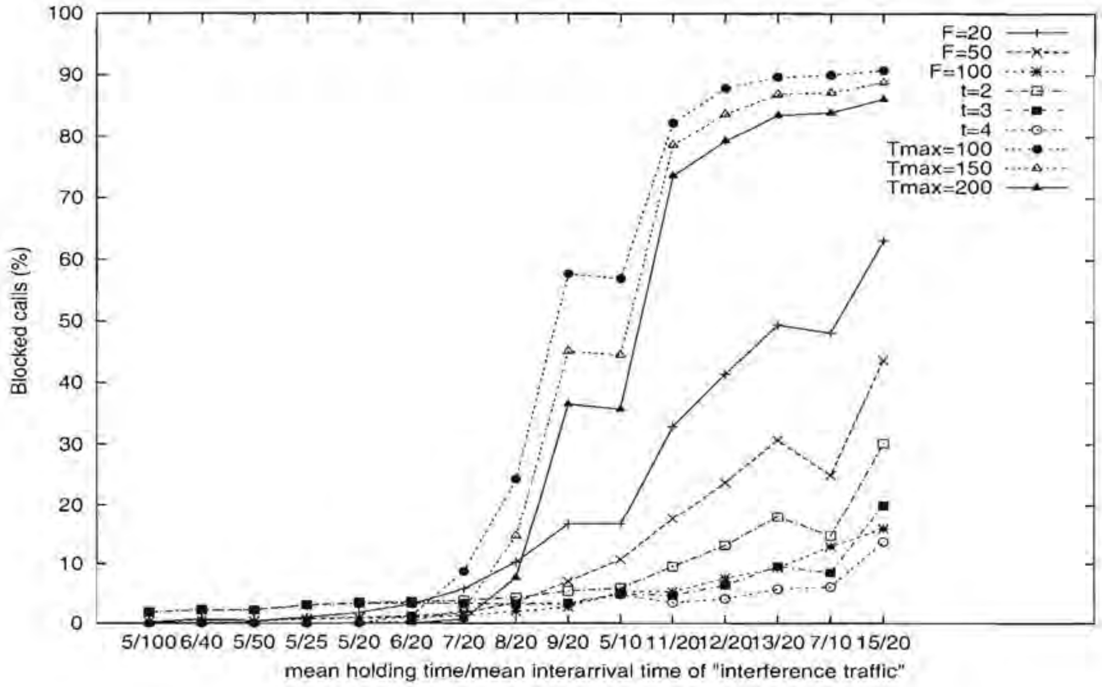


Figure 5.18: Percentage of calls blocked using different connection setup schemes

Figure 5.18 shows the percentage of calls blocked when each connection setup scheme is used. The $kT_{wait} - T_{max}$ scheme performs the worst, even with the usage of a D_{max} value of 200 seconds the blocking percentage is almost 90%. The reason for this high blocking rate can be explained by looking at the queue at *Switch1* in which connection requests requiring access to the channel that connects *Switch1* to *Switch2* are placed. With an increase in interference traffic, this queue grows very large and causes connection requests to experience delays larger than the D_{max} value used, causing the connection to be blocked. Due to the poor performance of the $kT_{wait} - T_{max}$ scheme (as shown in Fig. 5.20) we dismiss it from further comparisons.

The call blocking percentages of the F scheme proves that increasing the value of F decreases the call blocking probability significantly. With the usage of a large F value by *Switch1*, the chances that a succeeding switch can admit a call during the same time *Switch1* is able to admit it increases significantly. Using a small F value (for example 20 seconds) when the average call holding time is large (13 or 15 seconds) serves to increase call blocking.

Increasing the number of time ranges used by the *timeslots* scheme decreases blocking probability significantly. Increasing the number of time ranges selected by the ingress switch improves the chances that an intermediate switch will find an overlapping time

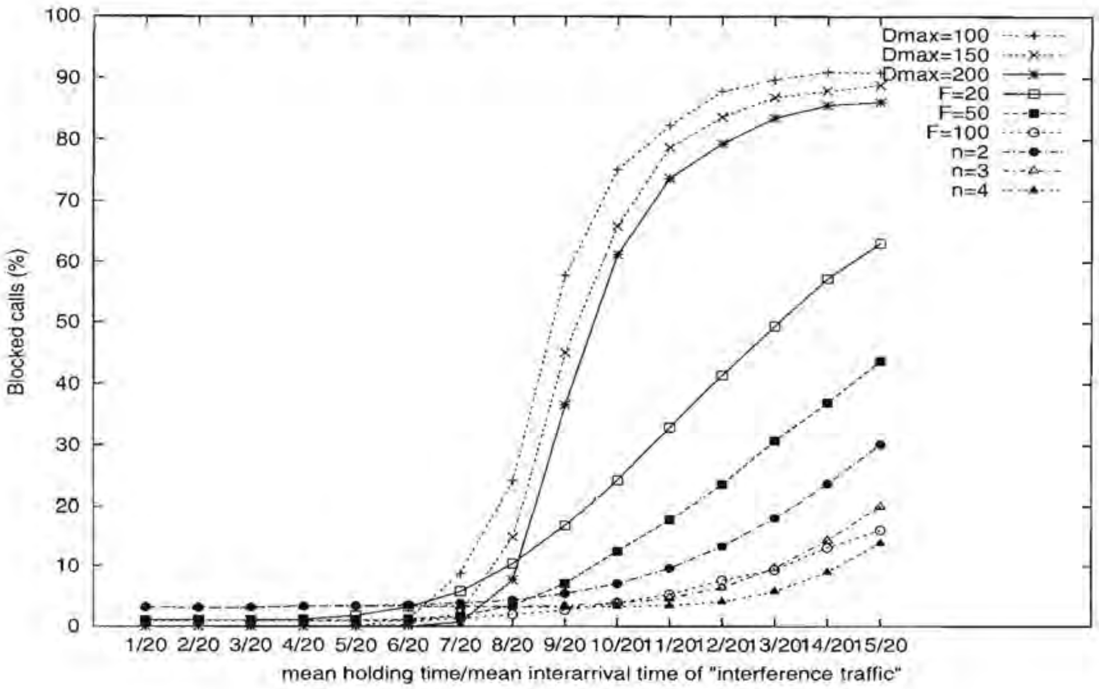


Figure 5.19: Percentage of calls blocked using different connection setup schemes, constant mean interarrival times

range during which it has the requested resources available.

The drop of call blocking probability at the points "5/10" and "7/10" in many of the graphs indicates the dependence of the various schemes on actual values of mean holding times and mean interarrival times. A rerun of the simulation where the interarrival times are kept constant, only varying the mean holding times confirms this. Figure 5.19 contains the results.

It is recognized that the blocking probabilities shown in Figure 5.18 and Figure 5.19 are rather high. By using much larger numbers of timeslots, and/or adding multiple path searches, the blocking probabilities can be reduced to more acceptable values than the currently shown 12% at 95% utilization for the $n = 4$ *timeslots* scheme.

The graph presented in Figure 5.21 shows, for all the remaining schemes, the difference in time from when *Source* requested a connection to *Dest* and the time when it is allowed to start data transmission for all successful connections. Using a large value for F in the F scheme has been proved to decrease the call blocking percentages, but now a disadvantage of a large value for F becomes apparent. Using a large value for F means that the host requesting a connection might receive a start time that is later than can in fact be handled by the network. This is because, in the case where $F = 100$ seconds, even if a connection

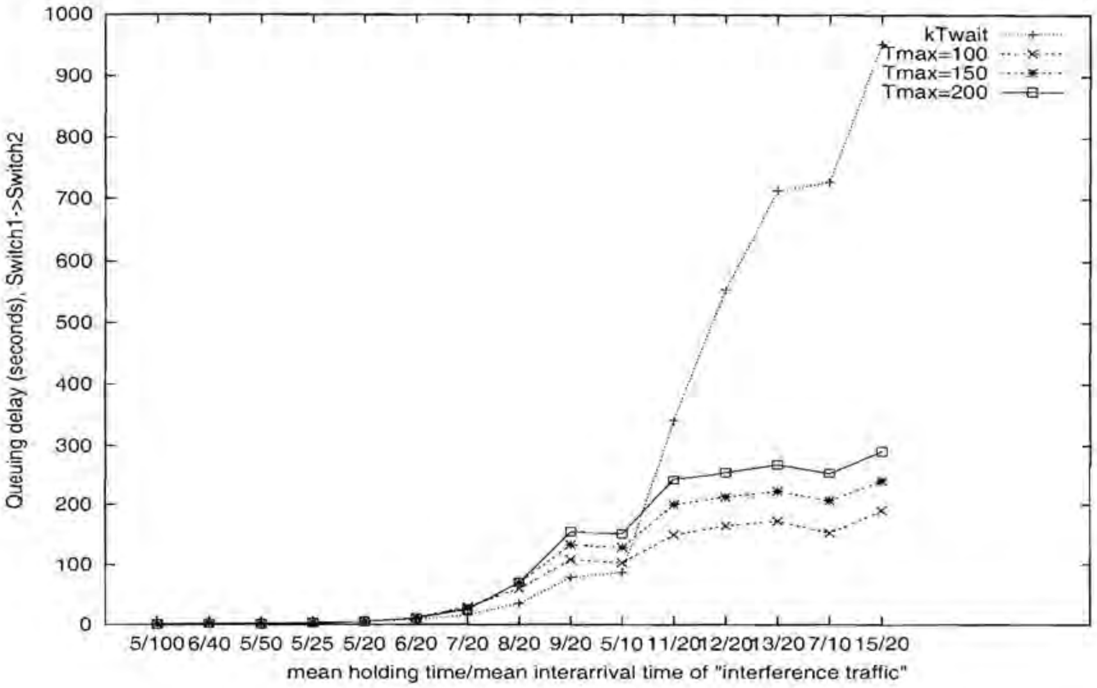


Figure 5.20: Queueing delay experienced by connection requests for connections requiring channel Switch1 -> Switch2

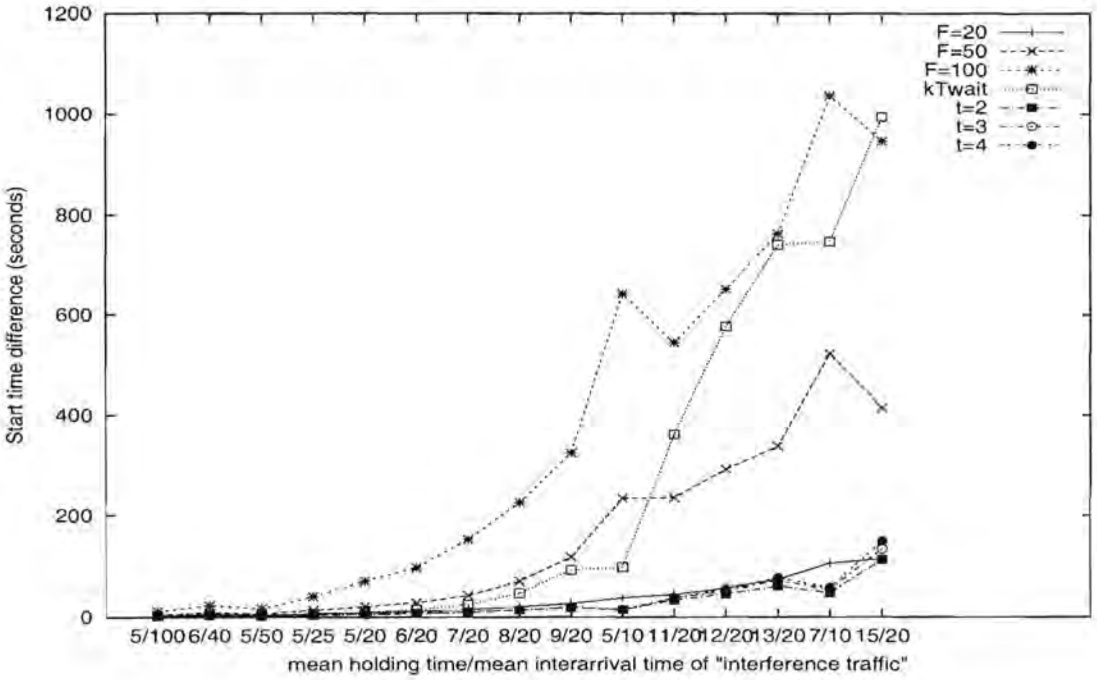


Figure 5.21: Start time delays for all connection setup schemes

will only be active for 5 seconds - the first switch will search for a time period of 100 seconds during which it has enough resources available to handle the connection. Even if the channel is free for 50 seconds when the connection request arrives - the connection will not be admitted immediately. This is the reason for the poor performance of the F scheme when a value of 100 seconds is used. Reducing the F value to 50 seconds and 20 seconds respectively reduces the start time delay considerably.

The *timeslot* scheme proves to be the best choice when a connection needs to be admitted as soon as possible. Note that the choice of different timeslots does not significantly affect the start time delay. This is because when the success reply is returned from the destination, it will always include the *epst* value as the start time of the earliest timeslot.

The kT_{wait} scheme's behaviour can be explained by looking at Figure 5.20. When the interference traffic contributes more than 50% to the total load, the queueing delay at *Switch1* grows exponentially. It appears that the kT_{wait} scheme is not able to handle a load of more than 70%. The advantage of the kT_{wait} scheme in that it does not block any connections now becomes insignificant. Even though the kT_{wait} scheme does not block connections, it becomes unstable when the load is higher than 70%. A source may be content to use the *timeslots* or F scheme to set up connections, and even if a connection request fails, the sending of a request more than once will result in less delays in these schemes than if the kT_{wait} scheme was used under high loads. Again, the dependence of the schemes on the actual values used for the mean interarrival time and mean holding time is illustrated in Figure 5.22.

Figures 5.23 and 5.24 represents the utilization of the OC1 channel between *Switch1* and *Switch2*. The *timeslot* scheme performs the best, even when the load of the network is at 95% (20% study traffic and 75% interference traffic), the first channel is utilized close to optimal. As expected, the F scheme performs poorly when a large value of F is used - although it still outperforms the kT_{wait} scheme. The moment the interference traffic passed the 50% mark, the kT_{wait} scheme's performance started dropping. The reason for this large drop is explained above.

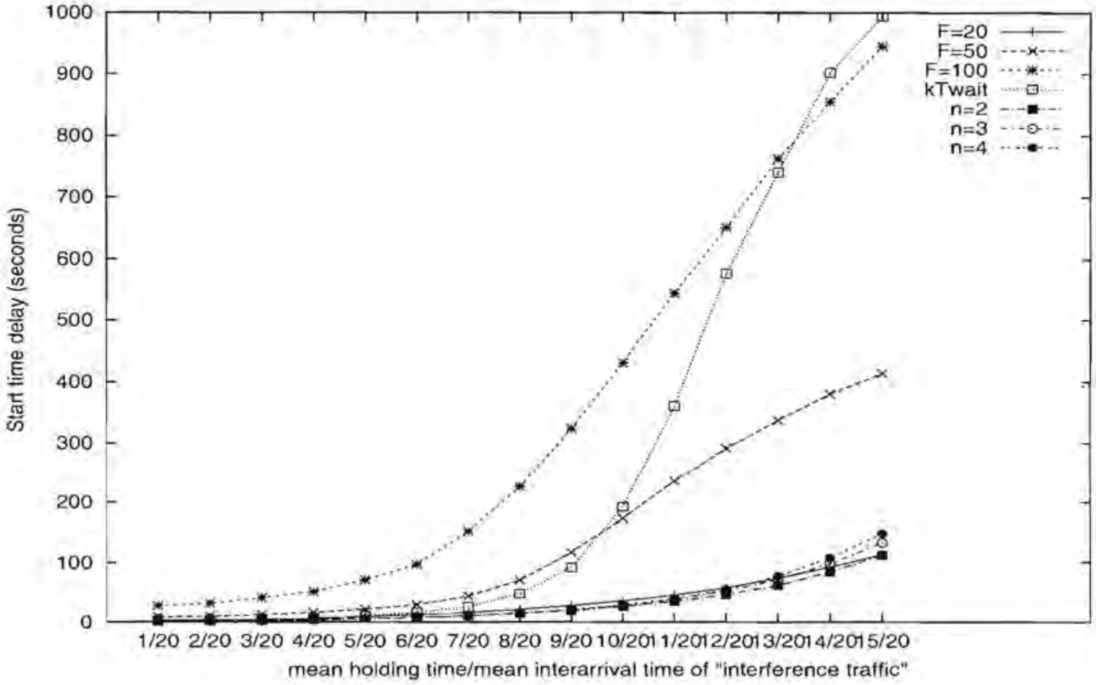


Figure 5.22: Start time delays for all connection setup schemes, constant mean interarrival times

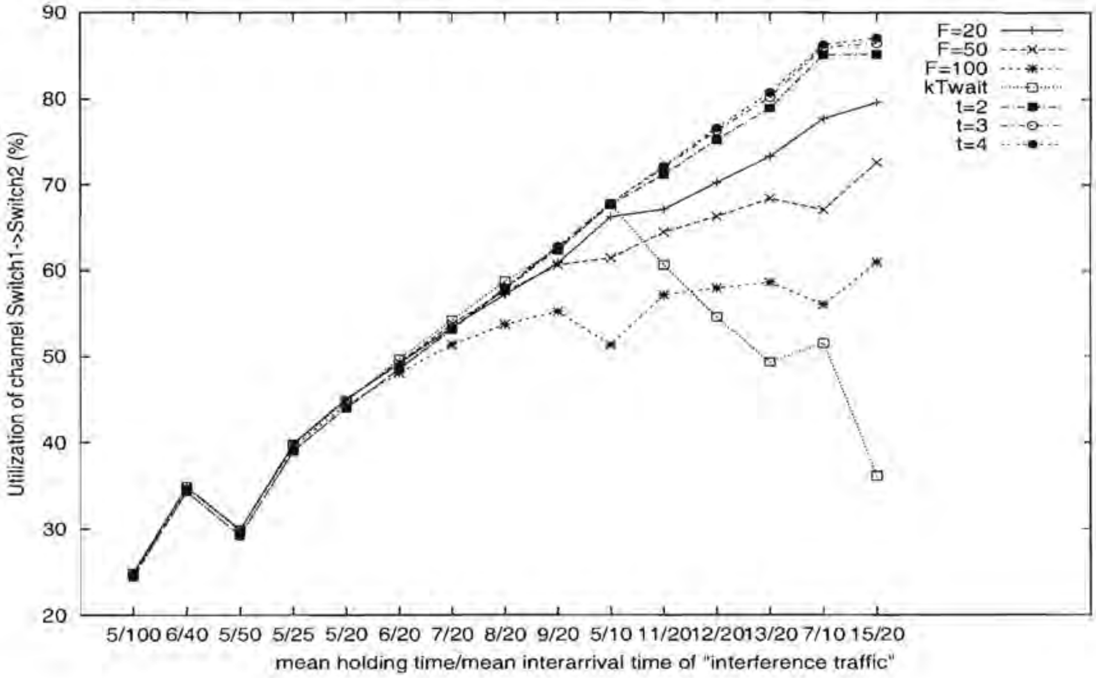


Figure 5.23: Utilization of channel Switch1->Switch2

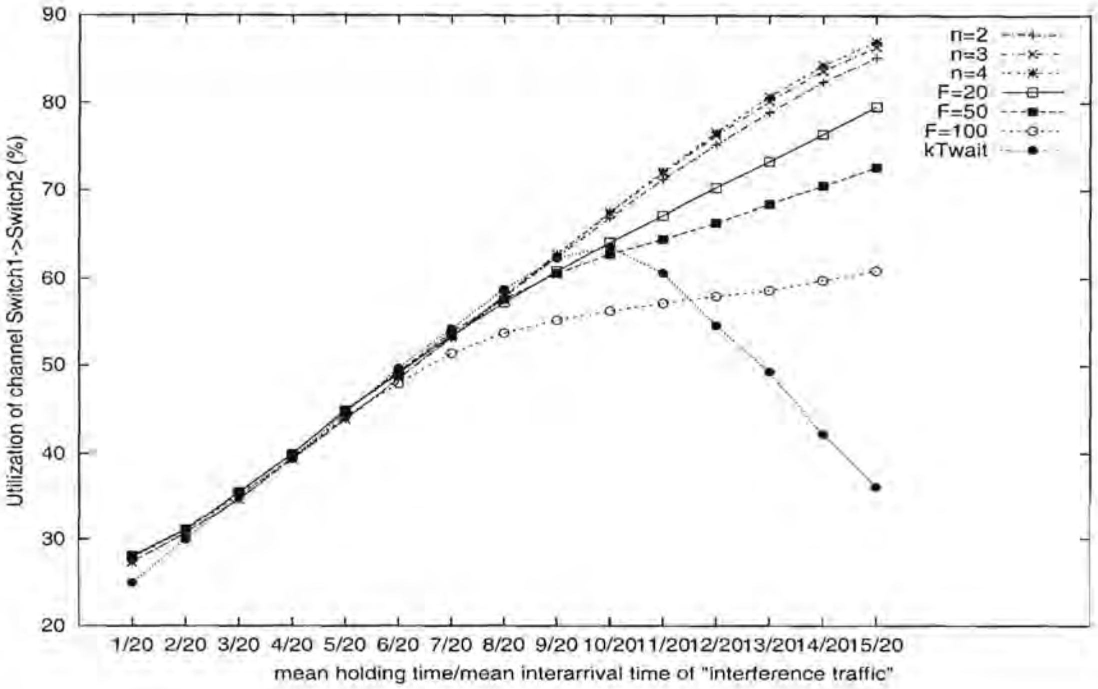


Figure 5.24: Utilization of channel Switch1->Switch2, constant mean interarrival times

5.7 Further extensions and future work

5.7.1 Switch programming delay

Switch programming is the act of setting translation or mapping tables at switches to realize circuits. In the case of circuit-switched connections (circuits), it involves the correct time slot/interface number or wavelength/interface number translations at the switches. In the case of virtual circuits as in ATM networks, it involves label mapping or virtual path identifier/virtual channel identifier mapping tables. The problem introduced by the equations discussed so far is that a switch forming part of the connection will initiate the programming of its fabric at exactly *epst*, which is the same time at which the end host (source) will start transmission. This may have the consequence that data from the source host may arrive at a switch before the resources have been reserved for it. This section considers two possible solutions to this problem.

Solving problem at switches

This solution assumes that each switch forming part of the connection has a value *prog_delay* associated with it, this is the value computed to be the delay from when a switch programming request is sent to the switch programming module until the switch fabric has been programmed with the information for the connection. Once the switch fabric has been programmed, the connection is active. Taking the value *prog_delay* into account, (5.1) changes to (5.11).

For each $j \in I$ determine

$$x_j \geq \text{ current time } \wedge a_j(t) \geq c, x_j \leq t \leq x_j + h$$

$$x \triangleq x_i \text{ where } x_i = min\{x_j \mid j \in I\} \; epst \leftarrow x + prog\_delay \tag{5.11}$$

This change to (5.1) can be applied similarly to all the relations and assignments discussed so far ((5.1) to (5.4)) replacing the variable *epst* with x, and computing *epst* with the assignment $epst \leftarrow x + prog\_delay$.

Solving problem at end host

Consider the value $prog\_delay(S)$, which indicates the delay introduced when programming the switch fabric of switch S. If the source host knows this value for all the switches along the path to the destination, it can compute the maximum of all these programming delays. Now the source host could solve the programming delay problem by only starting its transmission at *epst+ maximum programming delay along path to destination.*

5.7.2 Propagation delay computation

In the introduction to this chapter, the first example application mentioned for scheduled connections was file transfers. In that discussion it was mentioned that the holding time of a connection used to transfer a file can be computed using the equation $h = \frac{f + overhead \times f}{rate} + T_{prop}$.

The problem with this equation is the question of how T_{prop} is computed. All connections are unidirectional, so there is currently no way for two nodes in the circuit switched network to compute the propagation delay between them. The proposed solution to this problem is to choose a value P which is either an educated guess of the value of T_{prop}, or some large value which should be larger than any propagation delay in the network. If a large value is chosen for P, the connection should be released by the sending host after transmission is completed, as opposed to the finish time being used by the switch for connection release. This makes the bandwidth used by the connection available to other connections immediately after release. Bandwidth will not be wasted. There will nevertheless be a time frame from the actual release time until time P during which no connections will be scheduled. This problem resulting from the inability to compute T_{prop} needs more attention.

5.7.3 Time

All the connection setup schemes discussed so far will only succeed in a network where all the end hosts and switches are perfectly synchronized to a common clock. In the simulation environment this was easily modeled using one global clock. However, for an implementation of the connection setup scheme, some mechanism of clock synchronization has to be implemented. An extensive survey done in 1995 reported that most machines using NTP (Network Time Protocol) to synchronize their clocks are within 21ms of their synchronization sources, and all are within 29ms on average [12]. The consequences of different times at switches have not been examined, but it is expected that the scheduling mechanisms will not perform well in such an environment. This is because situations may arise when data is received before a connection is set up, or a connection is closed before data transmission is completed. Mechanisms that use relative time values or some such approach are needed to deal with these time differences.

5.8 Summary and Conclusions

This chapter described various scheduling algorithms for calls with known holding times. To support multiple-switch connections, two scheduling alternatives, the F scheme and *timeslots* scheme, were proposed. In the F scheme, resources are reserved for a long period F, longer than the call holding time, to guarantee that all switches would find a start time for the connection within this long period. In the *timeslots* scheme, multiple time ranges when the ingress switch can support the call are selected and sent downstream in the signalling message. Each intermediate switch then selects a subset from within this set, until the destination is reached. These two schemes were compared against a simple call queueing scheme, the kT_{wait} scheme, which does not assume knowledge of call holding times, and a variant of this scheme. A simulation study of the schemes showed that (a) significant gains are possible by using knowledge of known call holding times, if that is available, and (b) of the two alternatives proposed for scheduling calls with known holding times, the *timeslots* scheme proved to be the better alternative. The kT_{wait}

scheme becomes unstable when the traffic load exceeds 70%. At 70% loading, the *timeslots* algorithm that uses knowledge of holding times can offer start time delays that are 85% smaller than with the kT_{wait} that does not assume knowledge of call holding times. Also, the *timeslots* scheme can be used with traffic loads of 95%. As a result, at 70% load a channel utilization increase of 37% is possible using schemes that exploit knowledge of call holding times, when compared to the kT_{wait} scheme. The reason we consider the *timeslots* scheme the winner over the F scheme, is that at large values of F, call blocking is low, but the start time delay is high, and with small values of F, the inverse is true. On the other hand, in the *timeslots* scheme, both the call blocking probability and the start time delay are relatively low for all loads tested. Its drawback over the F scheme lies in the increased signalling overhead, which might be negligible if few time slots are used.

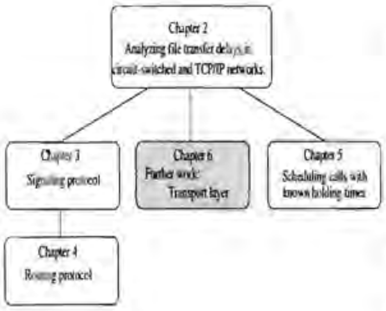
The novelty of this work lies in its introduction of call queueing and providing delayed starts for connections. It culminates in a proposal to use known call holding times while scheduling to improve both network resource utilization and call queueing delays.

Future work includes the incorporation of the *timeslots* scheduling scheme into the signalling protocol presented in Chapter 3.

Chapter 6

Further work: The transport layer

6.1 Introduction


Chapter 2
Analyzing file transfer delays in circuit-switched and TCP/IP networks.
Chapter 3
Signaling protocol
Chapter 6
Further work:
Transport layer
Chapter 5
Scheduling calls with known holding times
Chapter 4
Routing protocol

To enable an application to transfer bulk data, for example large files, over a circuit switched network, a transport layer is required to provide the application with the guarantee of a complete and error free transmission. The details of this discussion are beyond the scope of this research, and are currently used as guidelines for future work.

The large bandwidths available in a SONET network means that the product of end-to-end delay and bandwidth is very large. The "bandwidth*delay product" measures the maximum amount of data in transit between two nodes. For example, if the transport protocol TCP is used to transfer data between two SONET nodes, the "bandwidth*delay product" is the buffer space required at the sender and receiver to obtain maximum throughput on the TCP connection over the path, i.e., the amount of unacknowledged data that TCP must handle in order to keep the pipeline full [25]. Although solutions have been proposed in [25] to increase TCPs performance in networks with high "bandwidth*delay products", it is still not efficient enough to be run over a circuit switched network.

NETBLT (NETwork BLock Transfer) [26] is a transport level protocol intended for the rapid transfer of a large quantity of data between computers. The protocol features also allow it to perform very well in networks with a high "bandwidth*delay product". NETBLT was initially designed to run over IP and therefor provides a service of connection setup between the communicating nodes using a two-way handshake. In a circuit switched network, the transport layer does not need to set up a connection between the transport entities due to the fact that a (network layer) connection has already been set up between the communicating nodes.

Instead of examining more transport protocols to find an ideal protocol to be used for file transfers in a circuit switched network, the functions of transport protocols are examined. Section 6.2 briefly discusses the transport layer capabilities that are envisioned to be ideal when a circuit switched network is used to transfer bulk data. These capabilities are currently being evaluated by the CATT research group for the design of a new transport protocol.

6.2 Protocol requirements

The transport protocol is responsible for guaranteeing a reliable data transport service between two nodes connected with a uni-directional (from the source to the destination) channel. A uni-directional link limits the communication between the two transport entities, but creating a bi-directional link between the transport entities will result in a waste of bandwidth. For example, the smallest channel available in a SONET network is OC1 (51.84Mbps), which, when dedicated to communication between transport entities for the whole duration of a file transfer, will be very wasteful. Recall that all the nodes forming part of the SONET network will have interfaces connecting them to the IP network. It is therefore proposed that all messages from the destination transport entity to the source transport entity be sent over an IP network. These messages are few during a file transfer. They should not add an excessive load to the IP network.

Figure 6.1 shows the paths TPDUs (Transport Protocol Data Units) will follow between two nodes connected by a network layer connection.

The services provided by a transport protocol are:

- *connection management*, which is a mechanism needed to maintain state information regarding the data transfer at the source and destination;
- *flow control* to avoid buffer overflows at the destination;
- *acknowledgements* received by the source that indicate the reception of data (successful or unsuccessful) by the destination;
- *error handling* that includes error detection and recovery schemes;

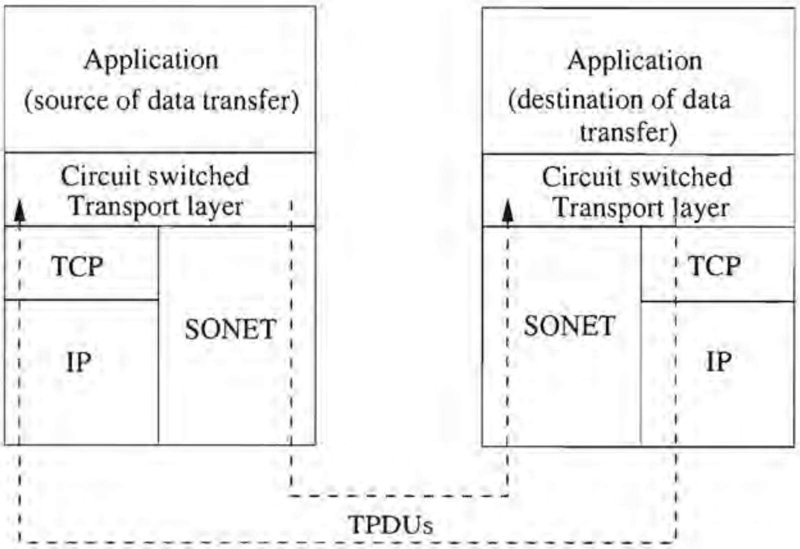


Figure 6.1: Transport layer in a connection oriented circuit switched network

- *congestion control* to handle lost packets in network routers.

Each service will be considered next, and recommendations are made for mechanisms that will be ideal for a transport protocol in a circuit switched network.

To transfer a file completely and without error, the two corresponding transport layers (at the source and destination) have to keep a record of which data has been received (at the destination), and how much data may be sent without acknowledgement (at the source). This is the *state information* of the transport protocol. The updates of state information, which should only be sent by the destination, will have to be sent out-of-band. That is, with the data being transferred over the circuit-switched network, the destination notifies the source about the progress of the transfer by sending messages over the connectionless packet-switched network (IP), as depicted in Figure 6.1.

It has already been suggested that there need not be a connection set up between the two corresponding transport layers because of an existing connection at the network layer level. The existence of a network connection allows the transport protocol to make use of an *implicit connection setup scheme* in which a transport layer connection is opened when the network layer connection is set up. The transport layer connection can now either be closed when the network layer connection is closed and/or the closing of the connection can be based on a timer when a scheduling mechanism similar to the ones discussed in Chapter 5 is used to set up connections.

The transport protocol has to bridge the gap between the services provided by the network layer, and the services required by higher layers [27]. In our target application of file transfer over a circuit switched network, it is assumed that when the (transport layer) connection is set up, the applications would already have agreed on the transfer rate acceptable to both the source and destination, which has been negotiated during the setup of the network layer connection. The decision to use an implicit transport layer connection setup scheme implies that all transport layer parameters that need to be negotiated are agreed to during network layer connection setup.

The rate at which the destination can accept data is taken into account when the circuit is set up. Because a circuit-switched network is used to transfer the data, the network will never become congested due to the reservation of resources in the network. But, once a connection is set up between the network layers of the source and destination, it is possible that, (due to variations in host tasks activity) the destination might not be able to accept data at the rate it had been able to handle when the network layer connection was set up. Thus, some *flow control* mechanism needs to be introduced to enable the destination to notify the source that it needs to limit the data transmission. For flow control, the mechanism of *rate control* is ideal. Using rate control the destination can ask the source to send data at a lower rate. For example, if an OC12 connection has been set up between the source and destination, the destination can request the source to send data at OC1 rate. This has the consequence that some resources might be wasted in the circuit-switched network during the times that the destination is not able to handle the initial data rate.

Data being transferred over a circuit-switched network will never be delivered out of sequence. This leaves the transport protocol with two responsibilities. First, all the data sent by the source has to reach the destination application free of bit errors (*error free delivery*). Second, all data transmitted by the source has to reach the destination (*complete delivery*). The usage of sequence numbers is required for error correction. Byte-based sequence numbers (as used in TCP) allows for efficient error correction by indicating the exact amount of data that is missing or containing errors, but considering the amount of bytes that could be in transit between two nodes in a SONET network, the sequence numbers will get really large. Thus a packet based sequence numbering scheme is proposed. Because the TPDUs will always arrive in sequence, a missing packet will be easily discovered. The destination can request the retransmission of a missing packet by sending a NAK (Negative Acknowledgement) to the source on the IP network. By making use of

NAKs, as opposed to positive acknowledgements (ACKs) only used for data received error free and in sequence, the signalling load is reduced considerably.

Additionally each TPDU could include a FEC (Forward Error Correction) code that is able to handle the most frequently occurring error patterns. When an error is encountered in a TPDU, a NAK is sent to the source requesting a retransmission of the packet (identified by its sequence number), similarly to the mechanism summarized in [28].

To determine if all the data transmitted by the source has reached the destination (*complete delivery*), it is proposed that all TPDUs are always the same length, except possibly the last TPDU. The last TPDU also includes an indication to report that it is the last TPDU of the transmission, thus indicating a complete transmission. Once the transport protocol receives this indication, it can signal the SONET signalling process to close the network layer connection. Given that circuits are best used with continuous traffic (rather than bursty), our thinking is that it is appropriate to disconnect the circuit when transmission is done.

6.3 Conclusion

Only a few design decisions have been considered for the transport protocol. The decisions should result in a light-weight transport protocol that allows for the complete and error free data delivery in a circuit switched network. The interfaces between the transport layer and the network and application layers still needs to be defined, as well as the detail of all the mechanisms described above.

Chapter 7

Conclusion

This research set out to find efficient mechanisms for bulk data transfer. An analytical and experimental analysis of file transfer delay in circuit-switched and TCP/IP networks suggested that circuit-switching is ideally suited for bulk data transfer.

To enable TDM networks to be used for bulk data transfer, a mechanism is required to enable the setting up of high-bandwidth on-demand circuits. This is because switched connections (as opposed to provisioned connections) are currently only available at the low rate of DS0(64Kbps) while data rates of OC1(51.84Mbps) to OC768(40Gbps) would be ideal for bulk data transfer. A signalling and supporting routing protocol that enables on-demand setup of circuits with rates of OC1 to OC192 was designed to reach this goal. For high throughput and a general increase in efficiency, the signalling protocol was designed to be implemented in hardware, and the routing tables used by it are updated by a routing protocol that was designed to be implemented in software. The signalling protocol is currently being implemented in hardware by the CATT research group.

Bulk data has the characteristic that when it is transferred over a connection-oriented network that implements preventative congestion control mechanisms (as opposed to reactive congestion control as implemented in TCP/IP networks), the duration or holding time of the connection can be determined by using the filesize, data rate and propagation delays.

Two scheduling algorithms were designed that use these known holding times. These algorithms enable a switch to reply with a later start time if it is unable to accommodate the connection request at the time of its arrival. The scheduling algorithms are simulated and compared to a simple queueing mechanism where a connection request is queued until its requested resources become available, only then being forwarded to the next hop of the connection.

The use of such scheduling is shown to improve the efficiency of bulk data transfer in connection-oriented networks. The simulations show that at 70% loading, these schemes offer start time delays that are up to 85% smaller and channel utilization of up to 37% larger than a simple queueing mode of operation where call holding times are ignored when connections are set up.

The evidence thus suggests that with appropriate signalling and scheduling, efficient bulk data transfer can be achieved on circuit-switched networks. Of course, a suitable transport protocol would be required to enable the reliable transfer of data between two hosts. While the ideal characteristics of such a protocol have been described in this dissertation, details of its design and implementation remain a matter for future research.