Chapter 1

Introduction

Based on a classification of the different networking and switching modes, three types of networking techniques are in existence today. A network can be packet-switched or circuit-switched and a packet-switched network can either be connectionless (CL) or connection-oriented (CO). Circuit-switched networks are, by their very nature, connection-oriented. In such networks, the switching action is based on the "position" of the arriving bits. "Position" is determined by the incoming interface number and time slot in a Time Division Multiplexing (TDM) switch, or wavelength in a Wavelength Division Multiplexing (WDM) switch. In CO packet-switched networks, packets that arrive on the connection after it has been set up carry the labels corresponding to the connection in their headers. When packets arrive at a switch in a CO packet-switched network, the mapping tables are consulted, labels are modified to corresponding values for the outgoing link (if needed), and packets are forwarded to the next node. No resources are reserved along the path between hosts communicating over a CL packet-switched network before data exchange is initiated between them. Each packet header contains the complete address of the destination and the packet is routed through the CL network based upon this information. Example networks from the above categories are presented in Table 1.1.

Switching mode \ Networking mode	Connection-oriented	Connectionless
Packet-switching	ATM,RSVP/IP,MPLS	IP
Circuit-switching	Telephony TDM network, SONET/SDH,WDM	

Table 1.1: Classification of networking techniques

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

Table 1.2: Classification of data transfers

Data transfers can be classified according to Table 1.2. The three categories where either the source or destination is "live" can be classified as "real-time" where the data is either sent live by the source or consumed live by the destination or both. Examples of these transfers include telephone conversations, telnet sessions, video and audio streaming (using compression techniques). These data transfers typically generate bursty traffic, have delay and jitter constraints, and endure for long sessions. For these applications the use of CO packet-switched networks are ideal because bandwidth is not wasted between bursts of data, which will occur when circuit-switched connections are used.

The remaining category involves file transfers. The data transferred is stored both at the sending and receiving end. Bulk-data transfers from applications such as web accesses, file transfers and electronic mail, could be "small" or "large." If the time to transfer a file is much larger than call setup delay, the file is classified as "large." Otherwise it is considered small. Small transfers are best handled by a CL network to avoid the overhead of call setup delay. Schwartz [5] (page 511), McDonald [6] (page 195) and Miyahara et al [7] shows that large bulk-data transfers are better handled in a circuit-switched mode than in a CL packet-switched mode. This is because the per-packet header and acknowledgment overhead in CL networks is greater than call setup overhead in circuit-switched networks. There is no intrinsic burstiness since a large file consisting of bits can be simply transferred across a network at a constant rate. This results in better performance (lower delays) when a large file is transferred over a circuit-switched network. Nevertheless, in practice, CL IP (TCP/IP) networks are mostly used for bulk data transfers. Using a CO packet-switched network for bulk transfer will suffer similar delays because the connection setup overhead together with the packet header and acknowledgement overhead will contribute to larger delays when transferring bulk data using a CO packet-switched network than transferring the same data using a circuit-switched network.

Two types of circuit-switched networks are in use today. Time Division Multiplex-

ing (TDM) is the simplest circuit-switched network where the switching action is performed based on the incoming interface number and time slot. Wavelength Division Multiplexing (WDM) switches base their switching decision on the incoming interface number and wavelength of the arriving bits. Currently, in both types of circuit-switched networks, on-demand (referred to as "switched" mode of operation as opposed to "provisioned" mode) circuits can only be obtained at the DS0 (64 Kbps) rate, and the only application that uses switched circuits is telephony. All higher rate circuits are used in provisioned (hardwired) mode where circuits are established a priori across the network and are changed infrequently.

To support large bulk-data transfers from applications such as web accesses, file transfers and electronic mail, a network will have to accommodate high-bandwidth (higher than DS0) on-demand circuits between any two nodes of the network. Considering that the high-speed circuit-switched network, Synchronous Optical Networks/Synchronous Digital Hierarchy (SONET/SDH), is able to provide circuits with bandwidth from OC1(51.84Mbps) up to OC768 (40 Gbps), there is a need for a mechanism to set up on-demand circuits for the benefit of increasing performance experienced by file transfer applications. Transferring a large file over even the least-bandwidth SONET circuit (51.84Mbps) will improve delays considerably when compared to the commonly used TCP protocol over the connectionless IP network.

In a CO network, connection setup and release procedures, along with the associated message exchanges, constitute the signalling protocol. Moving away from the telephony application, a new signalling protocol is required for the circuit-switched networks to support these high-bandwidth on-demand circuits.

Signalling protocols are typically implemented in software. This is due to the complexity of signalling messages, and the amount of state information that needs to be kept by all switches participating in connection setup. The fact that signalling protocols are implemented in software points to a shortcoming in circuit-switched networks. Switch fabrics are continually improved to handle data at faster speeds, but that is only after a connection has been set up. Implementing signalling protocols in hardware could significantly increase switches' call handling capabilities. The expected increase in call handling capacities of switches will dramatically affect the role circuit-switched networks play in future data transfer applications. This dissertation presents a novel signalling protocol intended for hardware implementation.

In support of the new signalling protocol, an accompanying routing protocol (to be implemented in software) is designed. The routing protocol provides enough information to the signalling protocol for efficient connection setup along the shortest path between the source and destination.

An experimental analysis of circuit-switching, done as part of this research, suggests that an improvement in network utilization can be expected based on an important characteristic of file transfer applications. The characteristic of interest is that if a file is transferred over a circuit-switched network (that implements preventative congestion control as opposed to reactive congestion control in TCP/IP networks), the holding time of the required connection can be deduced from the file size. In connection-oriented networks, connections are admitted without knowledge of the call holding time. If resources are not available, the connection request is simply denied and connection release procedures are initiated by the signalling protocol. This dissertation explores the option of queueing calls in connection-oriented networks instead of blocking them when network resources are unavailable. A simple call queueing algorithm is to hold up call setup messages at each switch along an end-to-end path until resources become available. This approach is used in the analysis described in Miyahara et al's paper [7]. This scheme suffers 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. Algorithms for such call scheduling are proposed and the quantitative benefits of algorithms that exploit knowledge of call holding times are demonstrated.

The author was a member of the CATT (Center for Advanced Technology in Telecommunications) research group at Polytechnic University, Brooklyn, New York, and is responsible for the design of the signalling protocol, the supporting routing protocol as well as the design and simulation of the scheduling algorithms. The scheduling algorithms presented in Chapter 5 has a US patent pending, reference number IDS 122874. Other CATT team members are currently exploring hardware implementation of the signalling protocol presented in Chapter 3, and busy designing a transport protocol to be used over a circuit for bulk data transfers.

Chapter 2 explores the performance gains that can be expected when a circuit-switched network is used for bulk data transfers. To provide a better understanding of the performance gains an experiment was designed to compare data transfer characteristics of

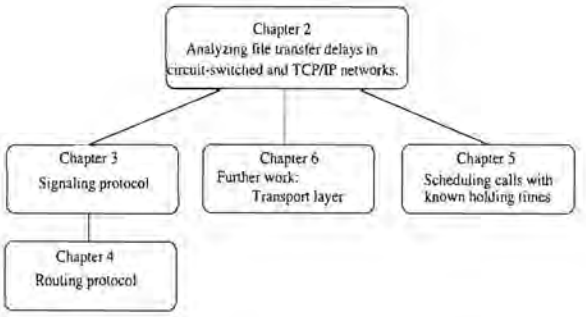


Figure 1.1: Chapter layout

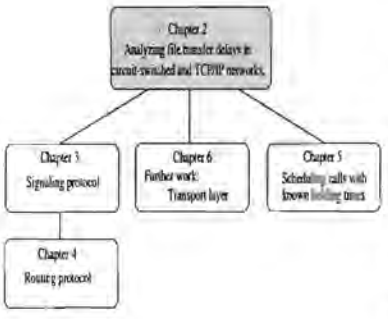
TCP/IP and circuit-switched networks. Chapter 3 introduces the signalling protocol for use in a connection-oriented circuit-switched network, in this case a TDM network, and Chapter 4 provides a solution for the supporting routing protocol. Chapter 5 describes some solutions that can be used to improve connection admission by making use of the holding time of the connection to schedule the start of a connection for a later time if the requested resources are not available at the time of connection request. Chapter 6 notes future work, which is the transport protocol that is required to enable the reliable transfer of data over a circuit-switched network.

Excluding the introduction and conclusion, the layout of the chapters can be presented with Figure 1.1. The conclusions drawn in Chapter 2 showed the need for a signalling protocol for circuit-switched networks (described in Chapter 3) and a transport protocol that enables the reliable transfer of data between a source and destination connected with a circuit-switched connection (discussed in Chapter 6). Chapter 5 investigates the queueing mechanism used in the comparison presented in Chapter 2. The signalling protocol requires a routing protocol to be able to set up a connection along the shortest path between a source and destination, this protocol is described in Chapter 4.

Chapter 2

Analyzing file transfer delays in circuit-switched and TCP/IP networks

2.1 Introduction



In theory circuit-switched networks have been shown to have lower response times for large data transfers than packet-switched networks [5] (page 511), [6] (page 195) and [7]. Since transferring a large stored file from one computer to another has no intrinsic burstiness associated with it, the total per-packet header and acknowledgement overhead in packet-switched networks is greater than the call setup overhead in circuit-switched networks.

To be able to transfer files over a circuit switched network, it is necessary that the network provide high-bandwidth on-demand circuits. We need high-bandwidth to be able to transfer large amounts of data quickly, and on-demand circuits are needed because these transfers could occur between any two nodes connected to the network. Currently in circuit-switched networks, connections of T1(1.5Mbps) and higher rates are set up in provisioned mode. This lack of flexibility might be the single biggest obstacle standing between circuit switching and its success in data networks. Indeed, currently we primarily use packet switching techniques like IP and ATM for the majority of network traffic, including large file transfers.

This chapter attempts to associate real world numbers with the comparisons in [5] and [7], starting by moving from general packet switching techniques to TCP/IP. TCP/IP is currently the packet switching technique of choice when large files are transferred over the Internet. Comparing TCP/IP to circuit-switching under laboratory conditions is intended to help us understand the performance we currently experience and to anticipate the performance we should expect if we decided to make use of circuit-switching for all large file transfers. Note that in this discussion, a large file is considered to be a file for which the total transfer time of the per-packet overhead and the acknowledgement overhead is greater than the call setup overhead in circuit-switched networks.

Due to the unavailability of a circuit-switched network that provides high-bandwidth on-demand circuits, we used UDP together with a simple network configuration to simulate circuit-switching. Section 2.2 describes the equations we designed for the comparison, Section 2.3 introduces the reader to the testing environment and the goals of the specific tests, Section 2.4 describes the results of the tests and Section 2.5 is the conclusion.

2.2 Comparing TCP/IP to circuit-switched networks

When transferring a file, delay can be categorized as follows:

- propagation delay
- emission delay
- queueing delay
- processing delay

In a large network with long round trip times, propagation delay will have a significant influence on the transfer time when TCP/IP is used. This is because of the presence of ACK messages. Before the maximum window size is used by the client, the propagation delay incurred by the ACKs will be significant.

Emission delays (time taken to put the bits on the wire) are also more significant when TCP/IP is used to transfer a file. When transferring data using TCP/IP, emission delays are present at the sending host and all the routers along the path. In the case of circuit switching, emission delays are only present at the sending host (see Figure 2.1). However, on closer examination, under optimal conditions pipelining could occur as will be explained below.

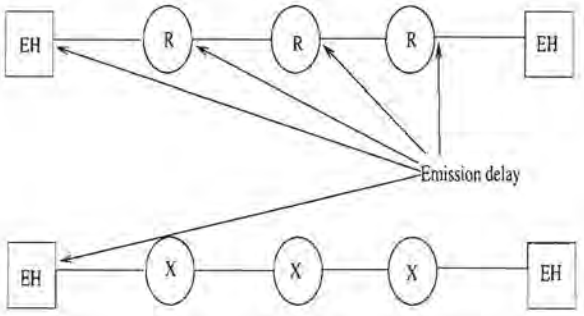


Figure 2.1: Emission delays, TCP/IP vs. circuit switching

The queueing and processing delays (experienced at routers when incoming packets are queued and processed) are also only present when sending the file via TCP/IP, not when the file is sent using circuit switching. Once a circuit has been set up between the source and destination hosts, the data is switched by each intermediate switch without it experiencing queueing and processing delays.

To better understand the total time taken to transfer a large file using TCP/IP vs. a fast circuit, we first propose mathematical models of the time delays in each respective case. Sections 2.2.1 and 2.2.2 present the mathematical models for circuit-switching and TCP/IP respectively.

2.2.1 Circuit-switched networks

$$T_{ckt} = T_{setup} + kT_{wait} + T_{transfer}(f) \tag{2.1}$$

$$T_{setup} = kT_{proc} + T_{r.t.prop} + 2(k+1)\left(\frac{s + ovhd}{rate}\right) \tag{2.2}$$

$$T_{transfer}(f) = \frac{T_{r.t.prop}}{2} + \frac{f + ovhd}{rate} \tag{2.3}$$

In the above equations, all the times (T) are the average times, s is the length of the signalling message, f is the file size, $T_{r.t.prop}$ denotes the round-trip propagation delay and k indicates the number of switches. T_{wait} is the time to wait for the requested resources to be freed (assuming a queueing mode of operation is used, as opposed to a call blocking mode). T_{proc} indicates the total processing delay that includes route determination, Connection Admission Control (CAC) and switch programming, incurred at a switch to set up a connection.

The overhead (*ovhd*) included in Equation (2.2) and (2.3) is the overhead introduced by the data link layer (for example SONET). If the transport layer implements a FEC (Forward Error Correction) scheme (supplemented with NAKs), the overhead will include more than just the headers added by the lower protocol layers.

Equation (2.1) is the total time needed to transfer a file using a circuit. Together with the processing delay (T_{proc}) experienced by a connection request captured in the term T_{setup}, the signalling message requesting the connection is also 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}.

Equation (2.2) includes the processing delay at switches, propagation delay, and emission delay (which is the length of signalling messages divided by the data rate used for signalling). It is assumed that a circuit is set up using two messages: typically a SETUP is sent in the forward direction from the source to the destination; if all the switches along the path and the destination are able to handle the connection, a SETUP SUCCESS message is sent in the reverse direction from the destination to the source. After processing each of these messages at each switch, a signalling message has to be re-introduced to the circuit-switched network. This has the consequence that the emission delay is computed for each switch (each switch places the signalling message "on the wire" to the next switch) as well as for the end hosts responsible for sending the message. Thus signalling messages have to be placed on $(k+1)$ links during connection setup in both the forward and reverse directions. Hence the term $2\,(k+1)$.

During file transfer (see Equation (2.3)), there is no queueing delay since these are circuit switches and only emission and propagation delays are incurred. The emission delay is also only incurred at the sending host. This is because after a circuit has been set up, the host initially places the bits on the wire after which it is switched by each intermediate node without experiencing queueing, processing or emission delays. It is assumed that the file transfer is initiated from the sending host, and the connection is closed by the sending host after the error free file transmission has completed. The consequence of this assumption is that the file transfer does not include a round trip delay, but only delay for transfer in one direction.

Time to release a circuit incurs the same three delays as to set up a circuit. When the sending host closes the connection, it sends a message with its request to its ingress

switch. After receiving a reply to this message, the connection is considered closed by the sending host. The total time for release is not directly included in the file transfer delay model given here, but the effect of release messages on queueing delays of setup messages should be considered.

2.2.2 TCP/IP

The following equations are obtained if it is assumed that there is no loss of packets and hence that the congestion window grows steadily with the sender sending 1 segment, and then 2, and then 4 and so on, until the whole file is sent. One ACK is assumed for each "group" of segments. It is also assumed that *ssthresh* is not reached and that TCP continues operating in slow start. In these equations k indicates the number of routers, m indicates the maximum segment size (in bytes) in TCP/IP and e-e indicates end-to-end.

$$T_{TCP} = T_{e-esetup} + T_{transfer}\,(f) + T_{e-eclose} \tag{2.4}$$

Equation (2.4) is the total time needed to transfer a file using TCP.

$$T_{e-esetup} = T_{e-eproc} + T_{r.t.prop.} + 3\,(k+1)\left(\frac{s+ovhd}{rate}\right) \tag{2.5}$$

Equation (2.5) represents the three-way handshake that is needed to set up a TCP connection. It includes the processing delay at routers (which includes queueing delay and service time), propagation delay, and emission delay. The term $T_{e-eproc}$ captures the total queueing and processing delay experienced by all the routers along the path to the destination. The emission delay is the byte size of signalling messages (s) divided by the data rate. In the case of TCP the signalling messages are all 40 bytes. The emission delay should also take into account the overhead added by the data link layer. For example, if Ethernet is used to transfer the messages, the overhead is 58 bytes.

$$T_{e-eclose} = T_{e-eproc} + T_{r.t.prop.} + 2\,(k+1)\left(\frac{c+ovhd}{rate}\right) \tag{2.6}$$

Equation (2.6) represents the closing of a TCP connection. Although there are more than 2 messages involved in closing a connection, this equation should typically only correspond to the last 2 messages. This is because only the last two messages do not contain data or acknowledgements corresponding to received data. The time delays caused by other segments responsible for closing the connection are included in the file transfer

time computation. The time taken to close a TCP connection incurs the same three delay components as were discussed in setting up a connection.

$$T_{transfer}(f) = NT_{r.t.prop} + T_{emission} + T_{total-svc} + T_{ack} \tag{2.7}$$

Equation (2.7) represents the total transfer time of a file of size f.

$$N = \left\lceil log_2\left(\frac{f}{m} + 1\right)\right\rceil \tag{2.8}$$

Equation (2.8) represents the number of groups of segments (N) needed to transfer a file of size f in segments that are m bytes in length. In the special case where the file size is such that each segment in each of the N groups is fully utilized, then: $f = \left(1 + 2 + 4 + 8 + 16 + ... + 2^{N-1}\right)m = \left(2^N - 1\right)m$. That is, $N = log_2\left(\frac{f}{m} + 1\right)$. In the general case the relationship $2^{N-1} < \frac{f}{m} + 1 \leq 2^N$ holds, justifying the ceiling function in Equation (2.8).

The first term in Equation 2.7 results from our assumption that one ACK is sent for each group of segments. Thus, a round trip propagation delay is incurred for each of the N groups of segments.

$$T_{emission} = \sum_{n=0}^{N-2}(k + 2^n)\left(\frac{m + ovhd}{rate}\right) + \left(k + 1 + \left\lceil\frac{f}{m}\right\rceil - 2^{N-1}\right)\left(\frac{m + ovhd}{rate}\right) \tag{2.9}$$

Equation (2.9) is obtained if we assume the emission delay on each link is pipelined. Per-link emission delay is the time to transmit a segment at the data rate *rate*. We first consider the emission delay experienced by groups in which each segment is fully utilized, that is, group x contains 2^{x-1} segments. For $n = 0, \ldots, (N-2)$, consider the $(n+1)st$ group. Each such group contains 2^n segments whose emission delay must be explained. Given that the emission delay experienced by one segment when it is placed on a link is $\frac{m+ovhd}{rate}$, the first segment of the group will experience $(k+1)\left(\frac{m+ovhd}{rate}\right)$ emission delay. If there are k switches on the path, there are $k+1$ links. The remaining $2^n - 1$ segments will each experience $\frac{m+ovhd}{rate}$ delay due to pipelining. Hence the term $(k + 2^n)\left(\frac{m+ovhd}{rate}\right)$ for emission delays.

The second term in equation (2.9) handles the emission delay experienced by the last group of segments that may not contain a number of segments that is an integer power of 2. The first segment of group N experiences $(k+1)\left(\frac{m+ovhd}{rate}\right)$ emission delay, the remaining number of segments, which is $\left\lceil\frac{f}{m}\right\rceil - 2^{N-1}$, each experience $\frac{m+ovhd}{rate}$ emission delay.

$$T_{total-svc} = \sum_{n=0}^{N-2} (k + 2^n - 1)\, T_{dp} + \left(k + \left\lceil \frac{f}{m} \right\rceil - 2^{N-1}\right) T_{dp} \tag{2.10}$$

Equation (2.10) is obtained if we assume the processing delay is pipelined at each router. In this equation the per-packet processing delay at routers, which includes queueing delays, is T_{dp}. Again, we first consider the emission delay experienced by groups in which each segment is fully utilized. For $n = 0, \ldots, (N-2)$, consider the $(n+1)st$ group. The first segment of each group takes kT_{dp} to reach the far end. After it has reached the destination, at the expiration of each T_{dp} one of the remaining $2^n - 1$ segments completes being processed. Hence the term $(k + 2^n - 1)\, T_{dp}$ represents the processing delays of group $(n+1)$. The first term is the sum of these delays for $n = 0, 1, \ldots, N-2$.

The second term of equation (2.10) corresponds to the processing delay experienced by the last group of segments that may not contain a number of segments that is an integer power of 2. The first segment of group N experiences kT_{dp} processing delay on its way to the destination. The remaining $\left\lceil \frac{f}{m} \right\rceil - 2^{N-1}$ segments each experience T_{dp} processing delay.

$$T_{ack} = N\frac{40 + ovhd}{rate} + NT_{queue} + NT_{svc} \tag{2.11}$$

The total delay experienced by ACK messages is given by Equation (2.11). For each group of segments, one ACK message is sent. This equation includes the emission delay, queueing delay and service time experienced by each ACK message traversing the network. The propagation delay experienced by ACKs are already included in the first term of Equation (2.7).

2.3 Laboratory experiment

This section describes an experiment run in a laboratory, measuring file transmit delays. Since the nodes were closely located in the laboratory, the influence of propagation delay on the overall transmission time was negligible. Instead, the emission and queueing delays were kept in mind when the network configuration was made.

Tests were conducted with a Sun SPARC4 workstation, an Intel PIII500 workstation and two Cisco 2500 routers, all connected with 10 Mbps Ethernet.

In all experiments the term "server" refers to the machine that sends a large file to the "client" that requested the file.

The network configurations presented in Figure 2.2 and Figure 2.3 were used.

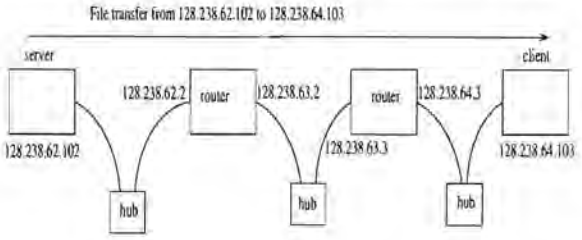


Figure 2.2: Network configuration used when transferring file with TCP/IP

The configuration used in the TCP/IP tests resembles a typical network. In this way the typical emission and low-load queueing delays were reproduced. Also, no extra load was introduced to this network. The only other traffic on the network was some negligible routing updates. This enables the measurement of the best case transfer time using TCP/IP.

UDP does not have the slow start and connection admission feature of TCP and consequently simply sends data at a constant rate. This enables us to emulate file transfer over a circuit (without the connection setup phase) by transferring a file using UDP. The network configuration used for the UDP test shows the "dedicated channel" between the sending and receiving host. Thus to emulate the absence of queueing delays in a circuit-switched network, we used a direct connection between the client and server. In our first experiment we realized the need for flow control with UDP. For this reason the client was the PIII500MHz machine receiving a file from the slower Sun SPARC4 - to prevent the client's buffers to overflow.

To measure the time taken to transfer a file with UDP, the client sent one ACK message back to the server to indicate that the file was received correctly. In all the tests, the times were measured at the server.

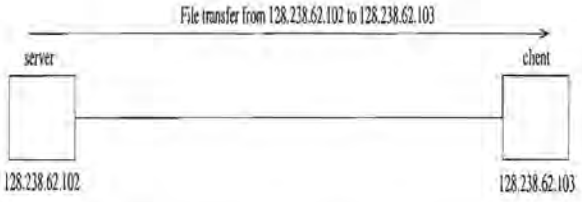


Figure 2.3: Network configuration used when transferring file with UDP

```
13:44:38.354435 128.238.62.102.32788 > 128.238.64.103.6234: S
        1569190891:1569190891(0) win 8760 <mss 1460> (DF)
13:44:38.354482 128.238.64.103.6234 > 128.238.62.102.32788: S
        169214293:169214293(0) ack 1569190892 win 32120
        <mss 1460> (DF)
13:44:38.356011 128.238.62.102.32788 > 128.238.64.103.6234: .
        ack 1 win 8760 (DF)
```

Figure 2.4: Establishing a TCP connection

2.4 Numerical results

In this analysis, the contribution of propagation delay is ignored in all experimental cases. Considering the network configuration used in these tests, with a typical distance of 7 meters between hosts, the propagation delay would be $\frac{7}{2.3*10^8} = 0.0304ns$, which is negligible in these tests.

Tests done with the help of the *traceroute* command found on most networked machines revealed that the average processing time for one packet in the end hosts is 0.4ms (this is the total processing time for both hosts). The average per-packet processing delay at each router (T_{dp}), which includes queueing delays, is 0.6ms.

2.4.1 TCP/IP

End-to-end call setup using TCP/IP is handled by a three-way handshake. Figure 2.4 shows message exchanges of a successful connection establishment between hosts 128.238.62.102 and 128.238.64.103 in *tcpdump* [1] format.

$$T_{e-esetup} = 3 \times k \times T_{dp} + 3 \times (k+1) \times \left(\frac{s + ovhd}{rate}\right) \tag{2.12}$$

By ignoring propagation delay from Equation (2.5), it can be changed to Equation (2.12). The processing and queueing delays experienced by the three connection-establishment segments is given by $(3 \times k \times T_{dp})$. The second term denotes the emission delay experienced by the three segments when they pass through a network with k routers $(k+1$ links). Our tests were run over an Ethernet network, introducing an overhead of 18 bytes. The message size in this case is 40 bytes, which is 20 bytes for the IP header and

```
13:44:51.427515 128.238.62.102.32788 > 128.238.64.103.6234: F
                9730049:9731181(1132) ack 1 win 8760 (DF)
13:44:51.427536 128.238.64.103.6234 > 128.238.62.102.32788: .
                ack 9731182 win 30987 (DF)
13:44:51.427775 128.238.64.103.6234 > 128.238.62.102.32788: F
                1:1(0) ack 9731182 win 32120 (DF)
13:44:51.429244 128.238.62.102.32788 > 128.238.64.103.6234: .
                ack 2 win 8760 (DF)
```

Figure 2.5: Closing a TCP connection

20 bytes for the TCP header.

After a file has been transferred with TCP/IP, the connection is closed gracefully. The server sends a FIN segment containing the last segment of data. This segment is ACKed by the client after which it sends a FIN segment of its own. On receipt of this FIN segment the server closes the connection, the client closes the connection after it receives the ACK for the FIN segment from the server. An example of this exchange of messages is shown in Figure 2.5.

The first two segments of the closing sequence will be handled by Equation (2.7). So, this equation is only applicable to the last two segments.

The propagation delay will be ignored, so the equation applied to our tests will be Equation (2.13).

$$T_{e-eclose} = 2 \times k \times T_{dp} + 2 \times (k+1) \times \left(\frac{c + ovhd}{rate} \right) \tag{2.13}$$

In Equation (2.13) the processing and queueing delay experienced by the last two segments is given by $(2 \times k \times T_{dp})$, and the second term denotes the emission delay experienced by the two segments when they pass through a network with k routers.

The TCP/IP implementation used for the tests implemented the ACK strategy of "ACK every other segment." This is different from our assumption used in Section 2.2, which stated that an ACK is generated for every group of segments. In none of our tests was delay introduced at the server due to a late acknowledgement. For this reason, the term T_{ack} shall be ignored in our analysis. This change to Equation (2.7), together with the removal of the term containing propagation delay produces Equation (2.14) in which

the emission and processing delays are represented. Equations (2.9) and (2.10) respectively model these delays.

$$T_{transfer}(f) = T_{total-svc} + T_{emission} \tag{2.14}$$

2.4.2 UDP as an emulator of the circuit switched mode

There is no connection setup phase in UDP, so the total transfer time can be given by Equation (2.15) which is similar to Equation (2.3) (the equation for the time taken to transfer a file of size f over a circuit switched network). The reason why the propagation delay is not halved to indicate the transfer time is because, in the UDP case, an ACK message is sent from the client to the server to indicate successful receipt of the data. The propagation delay will still be ignored due to its insignificant size, so the equations can be thought of as exactly the same in the laboratory environment. The second term of Equation (2.16) shows the ACK that is sent from the client to the server to indicate successful receipt of the file.

$$T_{transfer} = T_{r.t.prop} + \frac{f + ovhd}{rate} \tag{2.15}$$

$$f = filesize + 512 \tag{2.16}$$

2.4.3 The experimental results

Figure 2.6 shows the results for different file sizes f given in Table 2.1, ranging from 13 to 25661582 bytes, and the corresponding experiments done to compare the predicted values against real world tests. In the TCP tests, the number of routers (k) is 2, the message size (m) is 1024 bytes and the overhead ($ovhd$) is 58 bytes (per message of size m). In the UDP tests, the overhead ($ovhd$) is 46 bytes per segment.

In the graph (Figure 2.6) the predicted values (using our mathematical models) for TCP/IP are also plotted. To predict the file transfer delay using TCP/IP, we attempted to provide bounds for the experimental results by computing the file transfer delay under the assumptions of ideal pipelining ("optimistic") and no pipelining ("pessimistic"). Both the optimistic and the pessimistic graphs were plotted using Equations (2.12), (2.13) and (2.14). The difference between the graphs is in how the emission and processing delays (the terms $T_{total-svc}$ and $T_{emission}$ from Equation (2.14)) are computed. In the optimistic graph the emission and processing delays are based on ideal pipelining and expressed by

File sizes (bytes)
13
1205
4820
5898
14460
24796
62039
332008
544541
1091306
3309132
5654721
6120664
9731180
12226702
15918652
25661582

Table 2.1: Values used for file sizes (f) in comparison of circuit-switching and TCP/IP

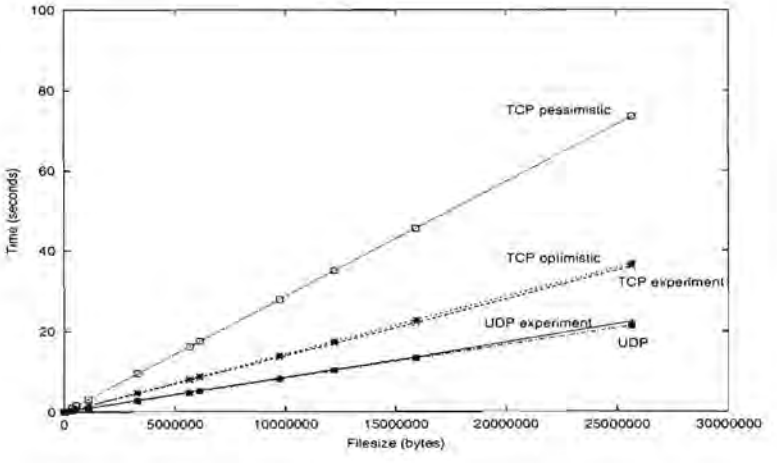


Figure 2.6: Comparison of transport protocols TCP and UDP

Equations (2.9) and (2.10) respectively. The pessimistic graph represents the emission and processing delays based on no pipelining as expressed by Equations (2.17) and (2.18).

$$T_{emission} = \left\lceil \frac{f}{m} \right\rceil \times \left(\frac{m + ovhd}{rate} \right) \times k \tag{2.17}$$

$$T_{total-svc} = \left\lceil \frac{f}{m} \right\rceil \times T_{dp} \times k \tag{2.18}$$

To see if UDP is a good emulator of circuit-switching Equation 2.15 was also plotted in Figure 2.6 (line "UDP").

A closer look at the TCP results suggests that the pipelining does occur. The "optimistic" graph presents a much better estimate of TCP performance than the "pessimistic" graph. Even if the TCP connection passes through k routers the dominant value is the emission delay on one link and the service time at one router, as anticipated in Section 2.2. The graph also indicates that UDP is a good emulator of circuit-switching.

The difference between the delays experienced with the two transport protocols is mostly due to the service time experienced by segments while passing through the network. For example, using Equation 2.14 and assuming ideal pipelining, the transfer of a file with a size of 12226702 bytes results in a total transfer time of 18.015 seconds of which 7.341 seconds is the service time (computed using Equation (2.10)). Using equation (2.15), the transfer of the same file will only take 10.221 seconds in the UDP network.

2.4.4 The analytical results

The laboratory did not allow much freedom to note the behaviour of the two transport protocols under operational conditions where large propagation delays might occur or a large number of routers might be in place (for analysis of emission delays in TCP/IP). Nevertheless, the experimental results from the previous section suggest that our equations provide a good estimate of the delay we can expect when a file is transferred using TCP/IP and a circuit-switched network. Thus, we continue to apply our equations with variables that indicate different network conditions and configurations. This allows us to compare the performance of TCP/IP to circuit switching more thoroughly.

The two network conditions of most interest are large propagation delays and large emission delays (large number of hops in a TCP/IP network). In a TCP/IP network, large propagation delays typically coincide with many hops. To divide the contributions

	Effect of propagation delay		Effect of emission delay	
Parameter	circuit-switching	TCP/IP	circuit-switching	TCP/IP
f	Table 2.1	Table 2.1	Table 2.1	Table 2.1
m	1024 bytes	1024 bytes	1024 bytes	1024 bytes
$ovhd$	4.4%	58 bytes	4.4%	58 bytes
$rate$	10Mbps	10Mbps	10Mbps	10Mbps
k	2	2	22	22
T_{proc}	100		100	
$T_{r\,t.prop}$	554.5ms	554.5ms	50ms	50ms

Table 2.2: Parameters used in analytical comparison of circuit-switching and TCP

made to the total delay by the propagation and emission delays respectively, two scenarios are explored and the behaviour of TCP/IP and circuit-switching in each scenario is examined. Firstly, a network with large propagation delays, but few intermediate hops was investigated. Secondly, a network with a large number of hops, but a relative smaller propagation delay was used.

The baseline for these analytical investigations was obtained from the following experiments. Testing with *traceroute* and *ping* revealed that the average round trip time between a certain machine in the USA and a certain machine in Europe is 554.5 ms. Another *traceroute* test between a machine in the USA and a machine in South Africa revealed a hop count of 22.

The same round trip time and number of hops is assumed in the circuit switching and TCP/IP computations. It is assumed that circuit switching is done with SONET (thus introducing a 4.4% overhead). The same rate of 10Mbps is also assumed in the formulae for computing circuit switching and TCP/IP times. It is assumed that 100ms of queueing and processing (parameter T_{proc} from Equation (2.2)) is required at each circuit switch to be able to set up a connection.

In each scenario the analysis of TCP was conducted for two implementations. In the first implementation there is only one ACK sent for each group of segments (using Equation (2.7)). The second implementation considered was the "ACK every other segment" strategy commonly used in TCP implementations. When making use of this strategy, the destination of a file transfer sends an acknowledgement for every second data segment received (except the fist data segment when the group's size is one segment, in which case

an acknowledgement is sent for one segment). This acknowledgement indicates the successful receipt of two data segments. Acknowledging every two data segments rather than every group of segments enables the source of the transfer to enlarge the group size while transmitting data, without waiting for an explicit acknowledgement that a group has been successfully received by the destination. Thus, once the group size is sufficiently large, the source will continue sending data without pausing to wait for an acknowledgement.

The "ACK every other segment" strategy is difficult to capture analytically. During the transfer of the first few groups, delays experienced by the ACK messages will not entirely be "absorbed" by the transfer of the actual data. A compromise was made to only include the last ACK message in the file transfer delay computation. That is Equations (2.7) and (2.11) were changed to Equations (2.19) and (2.20) respectively. This is just an estimation of the transfer time until a more appropriate equation can be found that captures the behaviour of the "ACK every other segment" strategy.

$$T_{transfer}(f) = T_{r.t.prop} + T_{emission} + T_{total-svc} + T_{ack} \tag{2.19}$$

$$T_{ack} = \frac{40 + ovhd}{rate} + T_{queue} + T_{svc} \tag{2.20}$$

Table 2.2 provides a summary of the parameters used in the analytical comparison.

Isolating effect of propagation delay

The propagation delay component can be isolated for the first scenario by combining the round trip time from the baseline with a small number of routers (hops). In this scenario the round trip time was chosen to be 554.5 ms and the value for k, the number of routers/switches was chosen to be 2.

Using Equation (2.3) for circuit switching, Equations (2.7), (2.9), (2.10) and (2.11) for TCP/IP with an "ACK every group of segment" strategy, Equations (2.9), (2.10), (2.19) and (2.20) for TCP/IP with an "ACK every other segment" strategy with the variables described above results in the graph presented in Figure 2.7. The interesting part shall be seen when we zoom into the graph where smaller files are concerned, which is shown in Figure 2.8. Looking at Figure 2.8 it is clear that TCP is much better suited for the transfer of small files, whereas circuit switching proves to be the network technique to use for the transfer of large files. It can also be seen that using the technique of "ACK every

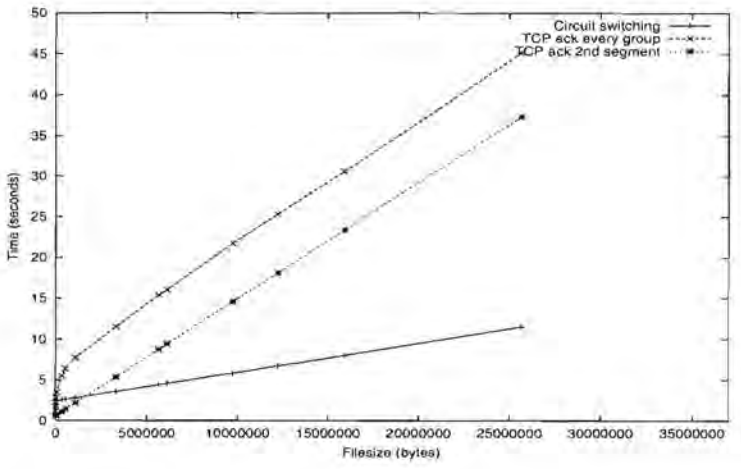


Figure 2.7: Effect of propagation delay on TCP and circuit switching

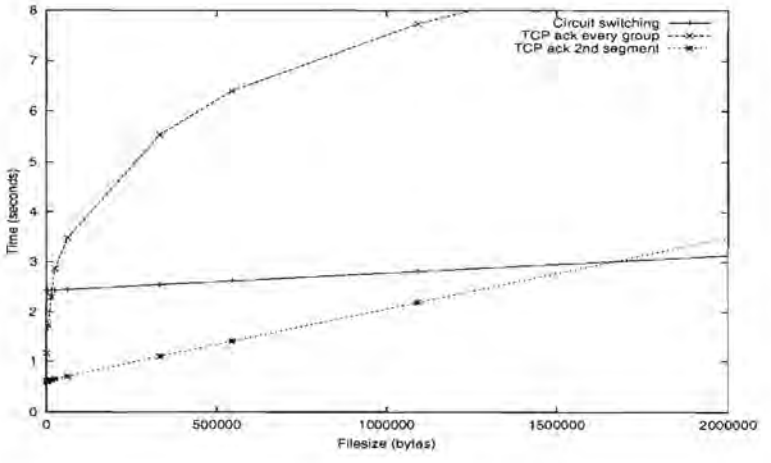


Figure 2.8: Effect of propagation delay on TCP and circuit switching, smaller files

other segment" seems to be more efficient than the "ACK every group" technique when a network with large propagation delays is used.

Isolating effect of emission delay

The emission delay component can be isolated for the second scenario by combining the number of routers (hops) from the baseline with a small propagation delay. The value 50ms was chosen for the propagation delay and the number of hops was chosen to be 22.

The same equations that were used in the examination of the propagation delay contribution are used with the new variables, and the results of this is shown in Figure 2.9. Again, a section of the graph is shown in Figure 2.10. The point at which the emission delay when using TCP/IP becomes more significant than the call setup delay in circuit

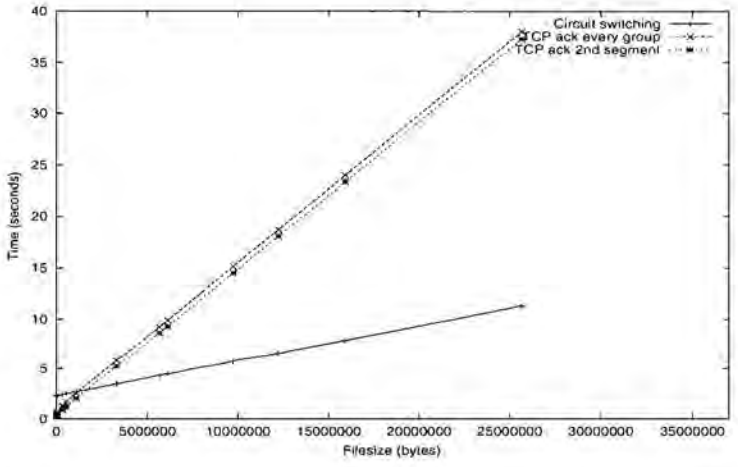


Figure 2.9: Effect of emission delay on TCP and circuit switching

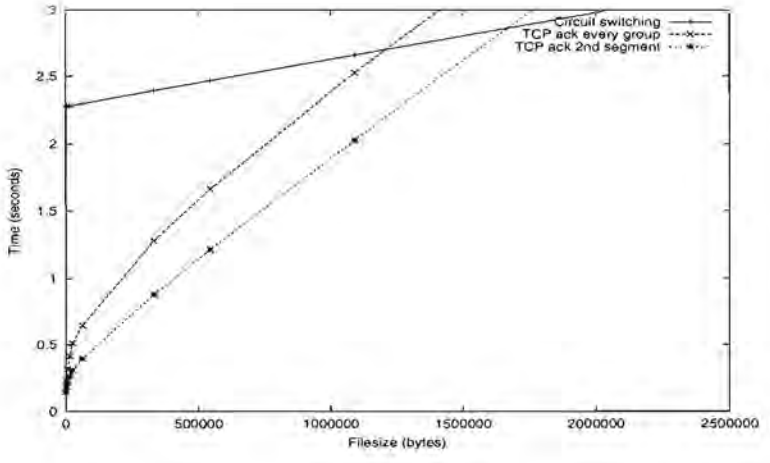


Figure 2.10: Effect of emission delay on TCP and circuit switching, smaller files

switching can be seen clearly in Figure 2.10. Sending any file larger than the file size at that point (just over 1GB) with circuit switching will result in smaller delays than using TCP/IP.

2.5 Conclusion

Queueing delays has changed over the years. In particular with the introduction of differentiated services it became necessary that all packet classification actions be performed at "wire speed" [8]. Thus, the processing of incoming packets has become fast enough to keep up with the rate of incoming packets. Queueing delay is instead incurred at the output port where the link insertion time and the number of packets destined for that link determine the queueing delay. An analysis of the contribution of queueing delay to the total delay is absent in the foregoing investigation. However such an analysis is unlikely to affect the broad conclusions to be drawn from the investigation. This is because the impact of queueing delay in circuit-switched networks is only present during connection setup. During data transfer, all data is switched without experiencing queueing delay. On the other hand, queueing delay will always contribute to the delay experienced by data transfers over CL packet-switched networks.

Thus the evidence we have seen generally suggests that circuit switching is better suited for large file transfers than TCP. Using TCP, emission, propagation and queueing delay all contribute significantly to the total delay when a large file is transferred . Propagation delay and call setup time does contribute to the total delay experienced when a large file is transferred using circuit switching. The result of interest is the fact that there is always a certain file size, that serves as the dividing marker between TCP and circuit switching. Transferring any file larger than this size using circuit switching will result in transfer speeds faster than if TCP was used.

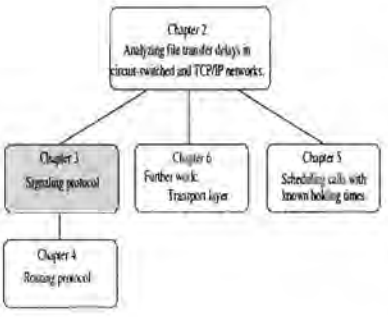
The need for transferring large files increases every day. Workstations are currently equipped with more storage space than would have been imagined a few years ago. There has also been an increase in the number of large file transfers over the internet. The increase is caused, for example, by the increase in occurrence of multimedia content available for free and by the availability of large software packages for download over the internet. Continuing this trend, the need to use circuit-switching will come to the fore more forcefully. However, as pointed out at the start of this chapter, circuit switching as a strategy can only succeed in an environment that provides high-bandwidth on-demand circuits.

Chapter 3 will propose a potential way of achieving this, based on appropriate original protocols.

Chapter 3

Signalling protocol

3.1 Introduction



Two types of circuit switched networks are in use today. Time Division Multiplexing (TDM) is the simplest where the switching action is performed based on the incoming interface number and time slot. Wavelength Division Multiplexing (WDM) switches base their switching decision on the incoming interface number and wavelength of the arriving bits. This work focuses on a TDM based circuit switched network.

Existing TDM circuit switched networks support the Plesiochronous Digital Hierarchy (PDH) up to the T3 (45 Mbps) rate and the Synchronous Optical Network (SONET)/Synchronous Digital Hierarchy (SDH) for higher rates up to OC768 (40 Gbps). Currently, on-demand (referred to as "switched" mode of operation as opposed to "provisioned" mode) circuits can only be obtained at the DS0 (64 Kbps) rate, and the only application that uses switched circuits is telephony. All higher rate circuits are used in provisioned mode. From Chapter 2 we note that for large file transfers, circuit switched networks that operate at higher data rates than the DS0 rate could be efficient to use. Also, since any end host can download files from any server, a switched mode of operation is necessary. These requirements introduce us to the need for a signalling protocol for high speed TDM networks that will enable them to offer the required switched mode of operation. This chapter discusses the design decisions made for such a signalling protocol and presents the protocol itself. Sections 3.2 to 3.4 describe the decisions made during the

design of the signalling protocol and an overview of the protocol itself. Sections 3.5 to 3.8 explain the protocol specification, and Section 3.9 introduces some future improvements to the signalling protocol.

3.2 Implementing the signalling protocol in hardware

One of the early decisions we made before designing the signalling protocol was to implement the protocol in hardware. Typically signalling protocols are implemented in software due to the complexity of the protocols and the need to keep the implementation flexible for evolution reasons. On the other hand, hardware implementations would yield a performance improvement so significant that the role and use of high speed circuit switched networks would change dramatically. To obtain such a performance gain while retaining some flexibility we recommend the use of reconfigurable hardware such as Field Programmable Gate Arrays (FPGAs).

Hardware implementations of networking protocols are on the rise. It has been quite common to implement physical-layer and data-link layer protocols in hardware. For example, Ethernet chips have been available for a long time for use in network interface cards. An Ethernet chip in a network interface card plugged into a host (workstation or pc) simply performs a 6 byte (MAC) address matching function. Chips are also available to perform Ethernet switching, where frames are switched on their destination MAC addresses.

Moving up the protocol stack, network-layer protocols are now being implemented in hardware. First, chipsets were created for ATM header processing as well as ATM switching. ATM is the network-layer protocol of a packet-switched Connection-Oriented (CO) network. In such networks, since a connection is first set up, the identifiers on which the switching is performed are simpler than in a packet-switched connectionless network-layer protocol, where switching is performed on destination addresses as in IP. More recently, network-layer protocols for even connectionless networks, such as IP, are being implemented in hardware. The throughput of IP routers (now called IP "switches") is greatly improved with hardware implementations. Also, the latency incurred per packet, which includes queueing and processing delays at each switch, is reduced. Thus there are many advantages to hardware implementations of networking protocols. Loss of flexibility as protocols are upgraded is cited as the main drawback of hardware implementations. However, now with reconfigurable hardware, this drawback should no longer be a serious

concern. It is also feasible to use hardware only for the basic operations of the protocol and relegate more complex and infrequent operations (for example, processing of options fields in the IP header) to software.

All of the protocols discussed above are used to carry user data, and are hence referred to as "user-plane" protocols. Our interest is in applying this same technique of improving performance through hardware implementations to "control-plane" protocols. Signalling protocols are control-plane protocols used to set up and release connections. These protocols are only needed in CO networks.

Current implementations of signalling protocols for both circuit-switched and packet-switched CO networks are done in software. Call setup signalling messages are queued at processors associated with switches and handled typically on a first-come-first-serve basis. Queueing delays are incurred under moderate and heavy load conditions. These delays add up across the switches dominating propagation delays and causing end-to-end call setup delays to be significant. Also, the processors handling signalling messages often become bottlenecks [2]-[4]. Hardware implementations of signalling protocols will greatly help improve both call setup delays and call handling capacities. By implementing the signalling protocol in hardware the throughput will increase significantly, thus a CO network will be able to admit more connections in a time period.

There are many challenges to implementing signalling protocols in hardware that have not been encountered in hardware implementations of user-plane protocols. The main difference is that in handling signalling protocols, the hardware needs to maintain state information of connections, whereas all the user-plane protocols implemented in hardware are stateless. Ethernet frames, ATM cells and IP datagrams are simply forwarded without any state being maintained for each packet or "flow".

The work items to provide a practical environment for such a signalling protocol include:

1. specification of a signalling protocol
2. implementation of the signalling protocol in hardware (using FPGAs)
3. design and analysis of how these signalling protocol chips can be used in a switch controller, and
4. design and analysis of end-to-end applications that can use high bandwidth on-demand circuits.

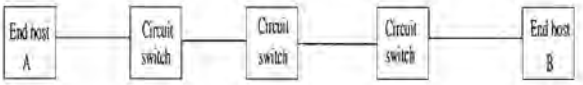


Figure 3.1: Ideal network in which signalling protocol will be used

This chapter only discusses the first work item, the specification of the signalling protocol. Work has already begun on prototypes of the hardware implementing a scaled down version of the signalling protocol and future work in this project shall deal with the other pending work items.

3.3 Mode of transport for signalling messages

The end goal of the signalling protocol is to be able to set up circuit switched connections on-demand. Before the specification of the protocol can be dealt with, one first has to consider the delivery of the signalling messages. Ideally the signalling messages should traverse the same type of network in which the connection is being set up. For example, in Figure 3.1 when end host A requests a connection to end host B, it should send its first signalling message requesting a connection (SETUP) to its ingress switch. In a circuit switched network, this means that there has to be a connection between end host A and its ingress switch. Also, for the signalling messages to traverse the network to the destination host (end host B), there has to be a dedicated signalling channel between all the switches and end hosts. To reserve such a channel in a TDM network for signalling results in a waste of valuable bandwidth. For example, if the circuit switch is a SONET/SDH switch with a cross connect rate of OC1 (51.84Mbps), the smallest channel available is OC1 which, when dedicated to signalling, would be very wasteful.

The proposed solution is to use another network as depicted in Figure 3.2. We propose carrying signalling messages on IP as out-of-band messages. Thus, when end host A wants to set up a connection to end host B, it sends a SETUP message to its ingress switch via its closest IP router. This has the consequence that routes and end hosts are identified by their IP addresses in signalling messages. Using IP to transport the signalling messages has two very important disadvantages: reliability and security. These issues will be considered under the improvements to the signalling protocol discussed in Section 3.9.

Using IP to send the signalling messages has a side benefit of reducing the size of the SETUP message by eliminating the need to include the source address in the SETUP

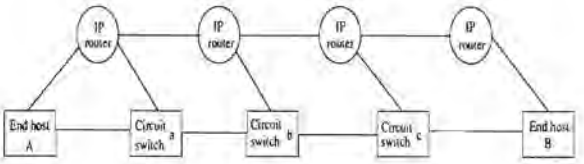


Figure 3.2: Typical network in which signalling protocol will be used

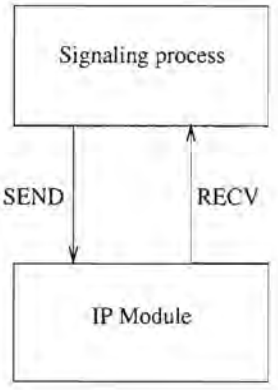


Figure 3.3: Primitives used between signalling process and IP module

message. IP provides the functionality of specifying source and destination addresses of the signalling messages in the IP header, the source address is needed at the receiving host to be able to reply to the connection request (SETUP SUCCESS). For example, when a connection is being set up between host A and B, the SETUP message sent from switch a to switch b in Figure 3.2 carries the source and destination addresses of end hosts A and B as parameters of the message, but carries the addresses of switch a and b in the IP header source and destination address fields. Each transit node can store the IP address of the previous node to use for the reverse SETUP SUCCESS message without it ever being included in the SETUP message.

The interface between the signalling process and the IP layer is implementation dependent. The following is from the IP specification [10]. "The functional description of user interfaces to the IP is, at best, fictional, since every operating system will have different facilities." Hence, the interface between IP and the signalling process should be developed according to the implementation. An example upper layer interface, consisting of two calls, that satisfies the requirements for the user (in this case the signalling process) to IP module communication is depicted in Figure 3.3.

The parameters of the calls are as follows ("=>" means returns) [10], the addition of the *SpecDest* parameter to the RECV call is in accordance with [11], which updates [10]

Parameter	Description	Length(bits)
src	source address	32
dst	destination address	32
prot	protocol	8
TOS	type of service	8
TTL	time to live	8
BufPTR	buffer pointer	Implementation dependent
len	length of buffer	Implementation dependent
Id (optional)	Identifier	16
DF	Don't Fragment	1
opt	option data	Implementation dependent
result	response OK = datagram sent ok Error = in arguments or local network	Implementation dependent

Table 3.1: Description of parameters in SEND primitive [10]

in this respect:

```
SEND (src,dst,prot,TOS,TTL,BufPTR,len,Id,DF,opt => result)
```

The description of the parameters in the SEND primitive is given in Table 3.1.

```
RECV (BufPTR, prot, => result, src, dst, SpecDest, TOS, len, opt)
```

The description of the parameters in the RECV primitive is given in Table 3.2.

Due to the freedom allowed in these two calls' specification, both have been changed to reflect the requirements of the signalling process. The changes are limited to the omission of unnecessary parameters. The *Id* is already optional, and not required by the signalling process and no optional data needs to be passed between the signalling process and the IP module. The length of the signalling messages (which are the only messages/data passed

Parameter	Description	Length (bits)
BufPTR	buffer pointer	Implementation dependent
prot	protocol	8
result	response OK = datagram received ok Error = error in arguments	Implementation dependent
len	length of buffer	Implementation dependent
src	source address	32
dst	destination address (may be broadcast or multicast address)	32
SpecDest	specific destination address	32
TOS	type of service	8
opt	option data	Implementation dependent

Table 3.2: Description of parameters in RECV primitive [10]

between the signalling process and the IP module) is either included in the signalling message itself, or it can be deduced from the message type. This eliminates the need for the IP module to include the length of the message when it is passed to the signalling process (the RECV primitive). The remaining parameters have the same description and length as before. The SEND call has been changed to:

```
SEND (src,dst,prot,TOS,TTL,BufPTR,len,DF => result)
```

The RECV call have been changed to the following:

```
RECV (BufPTR, prot, => result, src, dst, SpecDest)
```

Other implementation decisions are:

- The Type of Service (TOS) parameter indicates to the IP network the quality of service desired for the data transfer. It shall always be set to 0001 0100 which indicates a request for low delay and high reliability when a signalling message is transferred.
- The *len* parameter will be a 16 bit value that is an exact copy of the message length information element in the SETUP message. For all the other messages, it shall contain the length of the signalling message.
- The usage of two destination address parameters (*dst* and *SpecDest*) in the RECV call is required due to the IP facility for broadcasting and multicasting. The specific-destination address (*SpecDest*) is defined to be the destination address (*dst*) in the IP header unless the header contains a broadcast or multicast address, in which case the specific-destination is an IP address assigned to the physical interface on which the datagram arrived [11]. No signalling message should be destined for a multicast group or sent as a broadcast message over the network. For this reason, the signalling message shall immediately be discarded when the two destination parameters are not equal.
- Use 8 bits for the result parameter, and the value 200 shall indicate success.
- The protocol value of 253 will be used to indicate the signalling layer. This value is currently unassigned according to [21].

- When data is passed to the IP layer, the TTL parameter will always be set to 16. This number indicates the maximum acceptable number of routers between TDM switches.

- The DF parameter shall always be set to 1, which indicates that this message may not be fragmented by the IP network. This is because the maximum size for the SETUP message is 407 bytes and given that the IP specification, [10], dictates that all hosts must be prepared to accept datagrams of up to 576 bytes, the message will never need to be fragmented by the IP network (even if a maximum sized header is included in the datagram).

3.4 Signalling protocol description

This signalling protocol is intended to be used in a circuit-switched CO network where connections are used for large file transfers, which have been proven to be the ideal application for high speed circuit-switched CO networks. Also, in a quest for a simple design of this protocol, a homogeneous network (where all the switches operate at the same cross connect rate) is assumed to be the target network. Changes to the signalling protocol when it is used in a heterogenous network will be discussed under the improvements to the signalling protocol in Section 3.9. With the above implementation considerations, certain assumptions can be made in the design of this protocol. They are:

- Only unidirectional two-party connections are allowed. The reason for this is that the primary application is large file transfers. Connections would then primarily be needed to transfer a large file from a server to a client.

- The setup requests are generated by the sending end of the unidirectional connections.

- The bandwidth requested in a SETUP message is either equal to, or an integral multiple of the (homogeneous network) switch cross connect rate. The bandwidth can be requested with the specification of a range that indicates minimum and maximum bandwidth requirements. Using these two parameters, the network attempts to allocate the maximum requested bandwidth. If any node along the path of the connection cannot provide the requested bandwidth, the highest bandwidth less than the maximum requested and larger or equal to the minimum requested is reserved.

Destination node address	Data rate OC1			...	Data rate 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

Table 3.3: Routing table

Again, the file transfer application permits this since a file can be transferred at any rate, unlike streamed video or audio where a specific data rate is required.

Connection setup consists of four steps:

- Route determination
- Connection admission control (CAC) and interface selection
- Time slot selection
- Switch fabric configuration

On completion of data transfer, corresponding release procedures are executed to free all resources reserved for the connection. These connection setup and release procedures, along with the associated message exchanges, constitute the signalling protocol.

There are two types of routing, hop-by-hop routing and source routing. In hop-by-hop routing each node determines the next-hop node to reach the destination. In source routing, the ingress switch determines the end-to-end route, which is then carried in the connection setup signalling message. Each intermediate node merely consults this parameter to determine the next-hop node to which to route the connection. In both cases, routes can either be computed dynamically when a connection setup message arrives or can be precomputed and stored in routing tables as shown in Table 3.3. Since we are targeting hardware implementation of this protocol, we select hop-by-hop routing and route precomputation. Due to the granularity of the rates available in a TDM network, the construction of the routing table is simplified in that next hop nodes need only be kept for each rate provided by the network.

This simplifies the *route determination action* performed at the node to a simple routing table lookup. It also simplifies the parsing of SETUP messages by not requiring the

Next hop node	Interface number	Total bandwidth	Available bandwidth	Cost

Table 3.4: Available bandwidth table including cost

hardware to process a source route parameter. The signalling protocol essentially trusts its associated routing protocol to ensure loop-free routing data. However, to prevent indefinite looping, we do provide for a Time-To-Live (TTL) field in the SETUP message requiring switches to decrement the TTL field at each hop, and to stop a SETUP request received with the TTL field equal to 1. Making use of route precomputation has the consequence that real-time parameters (for example, the current load in a node) are typically not taken into account. This means the next-hop node according to the routing table might not have resources available for a connection at the time when the connection request arrives. This introduces the need for more next hop options which is included in the routing table as shown in Table 3.3. Including more than one "next hop" allows a node to send a request for a connection to an alternative node in the case when a previously chosen "next hop" does not have enough resources available.

Connection admission control (CAC) in a TDM switch consists of determining whether there is sufficient available bandwidth for the connection on any of the interfaces to the next-hop node identified in the route-lookup step. The connection admission control action is performed by examining the available bandwidth column of a table such as the one shown in Table 3.4, or by using some other connection admission mechanisms as will be discussed in Chapter 5. In this table a record of available bandwidth is kept on a "per-interface" basis. Signalling protocol hardware reads and writes data into the "available bandwidth" column of this table as it admits and releases calls. The other columns are written by the node administrator during configuration, except the "cost" column that will be discussed in Chapter 4.

The incoming interface numbers are obtained from information in the SETUP message. As connection setup proceeds hop-by-hop, each switch places its outgoing interface number as a parameter in the SETUP message. Assume node I receives a SETUP message from its neighbour I-1 indicating an outgoing interface number O. The interface number O corresponds to an interface on node I-1. Node I needs to determine the number of its interface(s) that is connected to interface O on node I-1. This is done by consulting a

Neighbour node		Interface number
Address	Interface number	

Table 3.5: Connectivity table

connectivity table, as shown in Table 3.5.

For example, interface number 5 on node I is connected to interface number 2 in its neighbour node I-1 as shown in Figure 3.4. If the SETUP message received from node I-1 indicates that the connection has been routed on its interface number 2, node I needs to use 5 as the incoming interface number.

Time slot selection is closely coupled with the interface selection that occur during CAC. Once it has been determined that there is enough bandwidth available on an interface, time slot selection will determine which time slots on the chosen interface will be used for data transfer. The time slots on an interface that will be used for the actual connection can be determined by the node during processing of the SETUP message, or the next-hop node may choose the time slots and provide the current node with the information in the success reply (SETUP SUCCESS) message. Whichever mechanism is used, the next-hop and the current node has to agree upon the time slots used for the connection before data transmission commence. With the introduction of maximum bandwidth selection, a node cannot know during processing of the SETUP message the exact amount of bandwidth that can be allocated by all nodes along the path to the destination. The bandwidth for a connection will only be known during the processing of the SETUP SUCCESS message. For this reason it is necessary for the next-hop node to include the time slots that will be used for the connection in the SETUP SUCCESS message.

One more benefit is received when the next-hop node is responsible for time slot selection. When time slot selection and/or switch configuration is done by a node upon receipt of a SETUP message the processing required when a connection has to be released is increased significantly. Keeping in mind our goal of hardware implementation, the processing in any node has to be kept to a minimum, which is accomplished by including the time slot numbers in the SETUP SUCCESS message.

Switch fabric configuration consists of writing data tables that map incoming interface and time-slot numbers to outgoing interface and time-slot interfaces as shown in Table 3.6.

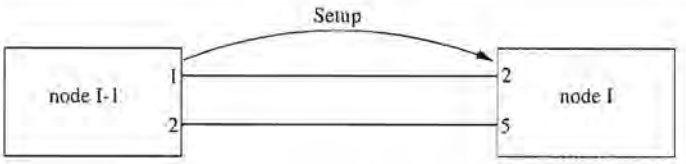


Figure 3.4: Connectivity information

Incoming channel identifier		Outgoing channel identifier	
Interface number	Time-slot number	Interface number	Time-slot number

Table 3.6: Switch mapping table

The connection setup actions, route determination and connection admission control/interface selection, occur sequentially as the setup procedure moves from node-to-node. Time slot selection is initiated upon receipt of the SETUP SUCCESS message, and switch fabric configuration is initiated either on a timer trigger, or with another message of which the SETUP SUCCESS message sent in the reverse direction is an example. When the SETUP SUCCESS message reaches the starting end host, it can send data on the newly established connection.

A connection could be released by the node that requested the connection (source), the node to which the connection was requested (destination) or any intermediate switch as a result of an error (for example, the physical connection breaks). At each node that forms part of the connection, the connection release procedure releases all resources reserved for the connection by editing the available bandwidth and switch mapping tables.

3.5 Signalling protocol specification

3.5.1 Messages

This section deals with the specification of the four messages of the signalling protocol, they are SETUP (Table 3.7), SETUP SUCCESS (Table 3.8), RELEASE (Table 3.9) and RELEASE CONFIRM (Table 3.10).

Information element	Length (bits)	Reference
Message type	4	Section 3.5.2.1
Destination address	32	Section 3.5.2.2
Source address	32	Section 3.5.2.3
Connection reference (previous)	12	Section 3.5.2.4
Time to live (TTL)	8	Section 3.5.2.5
Min bandwidth	8	Section 3.5.2.6
Max bandwidth	8	Section 3.5.2.6
Interface number	8	Section 3.5.2.7
Checksum	16	Section 3.5.2.8

Table 3.7: SETUP

Information element	Length (bits)	Reference
Message length	16	Section 3.5.2.9
Message type	4	Section 3.5.2.1
Connection reference (previous)	12	Section 3.5.2.4
Connection reference (own)	12	Section 3.5.2.4
Bandwidth	8	Section 3.5.2.6
Time slot number(s)	8-1536	Section 3.5.2.7
Checksum	16	Section 3.5.2.8

Table 3.8: SETUP SUCCESS

Information element	Length (bits)	Reference
Message type	4	Section 3.5.2.1
Connection reference (own)	12	Section 3.5.2.4
Cause	8	Section 3.5.2.10
Checksum	16	Section 3.5.2.8

Table 3.9: RELEASE

Information element	Length (bits)	Reference
Message type	4	Section 3.5.2.1
Connection reference (own)	12	Section 3.5.2.4
Cause	8	Section 3.5.2.10
Checksum	16	Section 3.5.2.8

Table 3.10: RELEASE CONFIRM

3.5.2 Description of Information elements

3.5.2.1 The *Message type* information element indicates the type of the received message. It can be any of four messages as depicted in Table 3.11. The protocol might be expanded to include more message types, which can be used for status enquiries, general notifications and updates. To support the scheduling scheme in Chapter 5 additional messages may be needed to trigger the programming of the switch fabric.

Bits 4 3 2 1	Message type
0 0 0 1	SETUP
0 0 1 0	SETUP SUCCESS
0 0 1 1	RELEASE
0 1 0 0	RELEASE CONFIRM

Table 3.11: Message types

3.5.2.2 *Destination address* is the IP address of the end host to which a connection is being requested.

3.5.2.3 *Source address* is the IP address of the end host requesting the connection.

3.5.2.4 A *Connection reference* can be assigned locally or globally. When only one node is responsible for the assignment of connection references that will be used by all the nodes forming part of a connection (*globally assigned*), complexity is added to the signalling protocol. For this reason, each node will be responsible to select a number that is used by the signalling protocol processing engines to keep track of connections. This number has local significance, changes hop by hop, and remains fixed for the lifetime of a connection. The connection reference is assigned by the

signalling protocol to identify the node forming part of the connection (as opposed to a link of a node). This is to enable the signalling protocol to be as light-weight as possible. By allowing the signalling protocol to select a connection reference from one range only (as opposed to a range associated with each outgoing link), its connection reference assignment is simplified. The SETUP message contains the *connection reference* being used by the node that sent the message. The RELEASE and RELEASE CONFIRM messages contain the *connection reference* being used by the node to which the message is sent. In reply to a SETUP message, the SETUP SUCCESS message to the previous node contains the *connection reference* of the node to which the message is being sent (which is the same as the one included in the received SETUP message), and the connection reference reserved for the connection by the node that sent the SETUP SUCCESS message. That is, the SETUP SUCCESS message sent from node i to node $i-1$ contains the connection references assigned by node i and node $i-1$.

3.5.2.5 The *Time to live (TTL)* information element is used to prevent indefinite looping. It is decremented at every node, and if it reaches the value 1 at any transit node (which is any node except the end nodes), the setup is deemed a failure.

3.5.2.6 The *Min bandwidth* and *Max bandwidth* information elements contain the minimum and maximum bandwidths requested for the connection in the SETUP message. The *Bandwidth* information element in the SETUP SUCCESS message contains the bandwidth agreed on for the connection. These information elements should contain the bandwidth in multiples of OC1 connections. Thus, when an OC48 connection is requested, the bandwidth (minimum and maximum) field shall contain "00110000". This allows for a request for a connection with bandwidth of up to OC192.

3.5.2.7 The *interface number* and *time-slot number(s)* information elements contain the values selected for the channels that will carry user data. Like connection references, these numbers also change at each hop. We allow for multiple time slot numbers in case the requested bandwidth is greater than the switch cross connect rate. Due to synchronization problems when one connection is handled by more than one interface, a connection is only able to be handled by one interface with the usage of one or more time slots.

3.5.2.8 A *checksum* information element allows for error checking on the received message. If the checksum fails, the message is immediately discarded by the signalling process.

3.5.2.9 The *Message length* information element is needed in the SETUP SUCCESS message because multiple time slots might be needed to accommodate the connection's requested bandwidth on the selected interface. The *Message length* information element indicates the number of 16 bit words of the message contents, that is, including the bytes used for the message length and the message type. If the message length cannot be expressed as a number of 16 bit words, padding may be used.

3.5.2.10 The *cause* information element indicates to the receiver of a RELEASE or RELEASE CONFIRM message the reason why the connection is being cleared. These values can follow the messages as defined in ITU-T Recommendation Q.2610. According to this recommendation, the information element has a variable length, but we shall only need the definitions of the different types of errors and their respective values.

3.5.3 Tables

The signalling protocol relies on the use of five tables, four of which have been discussed in Section 3.4, they are:

1. Routing table, Table 3.3;
2. Available Bandwidth table, Table 3.4;
3. Connectivity table, Table 3.5;
4. Switch mapping table, Table 3.6; and
5. State table, Table 3.12.

The state of every connection being established, released, or in use is maintained both at the end hosts and at the switches. The state diagram for the signalling protocol is as shown in Figure 3.5. In the *Setup received* state, two of the connection setup actions are performed. The last two connection setup actions, *time slot selection* and *switch programming*, is performed upon entering the *Established* state. In the *Release received* state, the release actions are performed.

Connection references			State	Bandwidth	Previous-node address	Next-node address	Outgoing channel identifiers	
Own	Prev node	Next node					Interface	Time slot

Table 3.12: State table

A state table such as the one shown in Table 3.12 is updated each time a signalling message is received. Since *connection references* are local, the connection references used by the previous and next nodes forming part of the connection are stored in the state table. The signalling protocol will always use the connection reference it assigned to the node (*own*) as an index to this table.

The *state* entry indicates the state of each connection with a number assigned to each of the states shown in Figure 3.5. The *bandwidth*, *previous-* and *next-node address* fields are maintained for each connection. The last column stores the *outgoing channel identifiers* of which there could be many per connection.

There are six states defined for this protocol. They are:

1. **Closed** No connection exits
2. **Setup received** This state exists in a succeeding node after it has received a SETUP message from the preceding node, but has not yet responded.
3. **Setup sent** This state exists in a preceding node after it has sent a SETUP message to the succeeding node, but has not yet received a response.
4. **Established** This state exists when a connection has been established.
5. **Release received** This state exists when a node has received a request from a preceding or succeeding node to release the connection, or when the higher layer (application) requests a connection release.

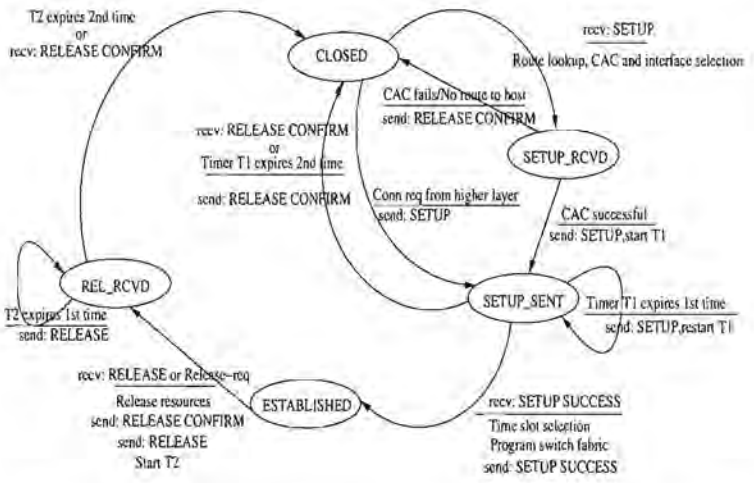


Figure 3.5: Signalling protocol state transition diagram

Figure 3.6: preceding/succeeding

3.6 Setup and release procedures for homogeneous networks

The setup and release procedures and the corresponding message exchanges are now briefly explained for switches setting up a unidirectional connection.

In these explanations preceding node refers to a network node that comes immediately before another network node in the connection setup path. Succeeding node refers to a network node that comes immediately after another network node in the connection setup path, see Figure 3.6. For clarity, the use of the IP network to carry signalling messages is ignored in this description.

3.6.1 Connection request

On receipt of a SETUP message from a preceding node, a node shall enter the *Setup received* state and execute all connection admission procedures. If there are not enough resources available for the connection, connection clearing procedures will be initiated by sending the message RELEASE CONFIRM to the preceding node with the "cause" field set to *bandwidth unavailable*. The node shall also return to the *Closed* state. If enough resources are available the node shall perform interface selection and the connection

establishment shall continue.

Connection establishment is initiated by the preceding node by sending a SETUP message to the succeeding node which is determined by a route table lookup. After sending the SETUP message, the preceding node enters state *Setup sent*. If, after a certain timeout, there has been no response to the SETUP message, the message may be retransmitted. After the second expiration, the preceding node enters state *Closed* and sends the message RELEASE CONFIRM to the node it received the SETUP message from with the "cause" field set to *timer expired*.

On receipt of the SETUP message, the succeeding node shall enter the *Setup received* state, and the above procedures are repeated at that node.

On receipt of a SETUP message, the node decrements the TTL. If the value results in 1, it shall enter the *Closed* state and send a RELEASE CONFIRM message to the node it received the SETUP message from. The "cause" field shall be set to *TTL expired*.

3.6.2 Connection establishment

Upon receiving a response from the application that the connection has been accepted, the destination node initiates the final phase of connection setup by choosing available time slot numbers on the incoming interface and sending the message SETUP SUCCESS to its preceding node. After this it enters the *Established* state. This message indicates to the preceding node that a connection has been established from its interface to the called party. On receipt of a SETUP SUCCESS message, a switch finalizes connection setup by performing the switch configuration step. The SETUP SUCCESS message propagates to the end host that requested the connection, which on receipt of the message, may start with data transmission.

Figure 3.7 shows an example of the message sequence when connection establishment is initiated by Host1, and is successful.

Figure 3.8 shows an example of the message sequence when connection establishment is initiated by Host1, but is rejected by Host2.

3.6.3 Clearing a connection

A call clearing request can be initiated by any of the two parties involved in data transmission. It is also possible for any node forming part of the connection to initiate call clearing due to a failure, administrative action or some other exception.

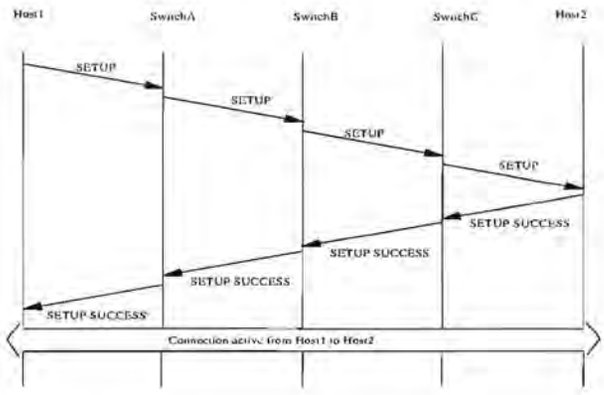


Figure 3.7: Successful setup of a connection

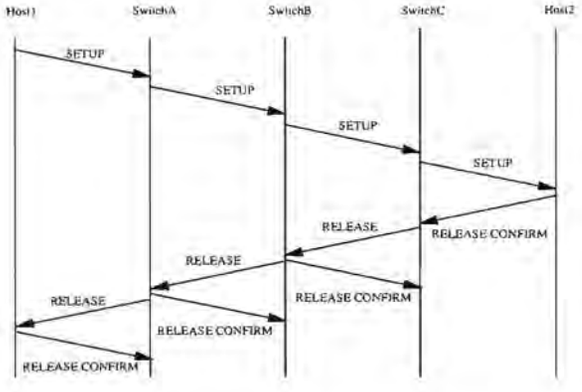


Figure 3.8: Unsuccessful setup of a connection

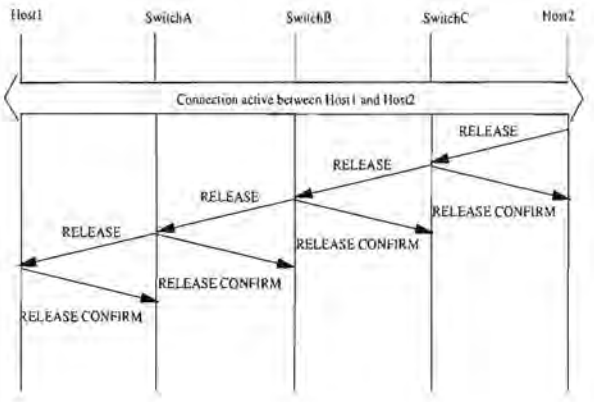


Figure 3.9: Normal connection release

Clearing the connection is initiated by the preceding node by sending a RELEASE message and initiate procedures for clearing the connection. These procedures are the notification of the application if appropriate, releasing resources held by the connection and clearing from all data tables the information of the connection. After this, it enters the *Release sent* state. In this state it expects the RELEASE CONFIRM message, after which it shall enter the *Closed* state.

Upon receipt of the RELEASE message, the succeeding node shall enter the *Release received* state and initiate the procedures for clearing the connection. After this, the succeeding node shall send a RELEASE CONFIRM message to the preceding node, send a RELEASE message to the next node in the connection and enter the *Closed* state.

Figure 3.9 shows an example of connection release initiated by the host that received the request for the connection.

Being a unidirectional connection, it is assumed that all connection releases will be initiated by the host receiving the request for a connection. This is because when a unidirectional connection is used to transfer a large file from a source to a destination, the destination is the only party that will know when data transmission has completed. Thus, if this end host receives a RELEASE message, there had to be an error in the network and the software should receive an interrupt indicating an error.

Figure 3.10 shows an example of message exchanges when an intermediate node requests the release of a connection.

During connection release it becomes apparent why the need exists to store three connection references in the state table. On receipt of a RELEASE message, a node has to determine which connection the message refers to. In the RELEASE message there are

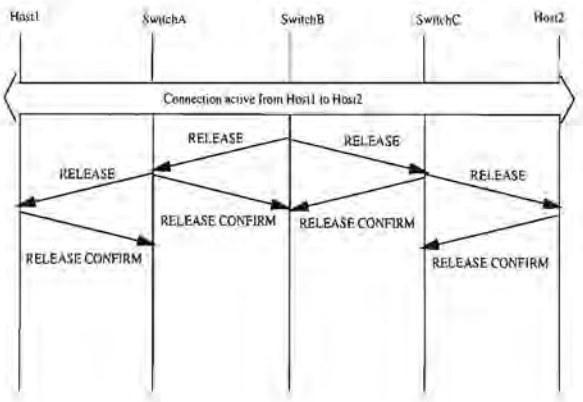


Figure 3.10: Connection release by an intermediate node

two indications, firstly there is the *connection identifier* in the message itself, secondly there is the *source address* which can be obtained from the IP header.

When a RELEASE message is received, the connection it refers to is found easily in the state table by using the connection reference included in the message as an index to the table. The source address (from the IP header) is then compared to the previous and next node entries in the state table to determine where the message came from. For example, if the message came from the previous node, a RELEASE message is sent to the "next" node containing the "Next node connection reference" and a RELEASE CONFIRM message is sent to the previous node containing the "Previous node connection reference."

3.7 Error conditions

The cause values and timers used by the signalling protocol are defined in this section. An 8 bit cause value is inserted in every RELEASE and RELEASE CONFIRM message to indicate the reason for failure. Depending on the cause value, the recipient of the RELEASE or RELEASE CONFIRM message may retry a connection request by altering the error causing parameters. For example, if a RELEASE CONFIRM message with a cause value of 1010 0010 (*Bandwidth unavailable*) is received in response to a SETUP message, the end host requesting the connection has the option of reducing the bandwidth requested and sending a new SETUP message.

This section also defines two timers that will be used in the signalling protocol.

3.7.1 Cause values

The cause values use the same format, and some values as specified in [13], although the definition and usage of the cause values may differ. This format for the cause values has also been used in [14] and [9].

In the RELEASE and RELEASE CONFIRM messages, the cause value field is 8 bits long, with the first (most significant) bit always set to 1. The cause value is divided into two fields, a class (bits 5 through 7) and a value within the class (bits 1 through 4). The cause classes are given in Table 3.13 and the cause values are given in Table 3.14.

Class	Nature of event
000	normal event
001	normal event
010	resource unavailable
101	invalid message
110	protocol error

Table 3.13: Cause classes

3.7.2 Timers

Whenever a signalling message fails a checksum test, it will be discarded immediately. There are no retransmission requests included in the current protocol. This, together with the usage of an unreliable network protocol (IP) to transfer the signalling messages prompts the requirement for some mechanism to handle erroneous or lost signalling messages. A solution is to make use of two timers in this signalling protocol, T1 and T2.

T1 is started whenever a SETUP message is sent. It is stopped when a SETUP SUCCESS or RELEASE CONFIRM message is received. On first expiry, the SETUP message is resent (and T1 is restarted). On second expiry, the connection is released with error *Nr. 11, Timer T1 expired* and the Closed state is entered.

T2 is started whenever a RELEASE message is sent. It is stopped when a RELEASE or a RELEASE CONFIRM message is received. On first expiry, the RELEASE message is resent (and T2 restarted). On second expiry, the connection is released with error *Nr. 12, Timer T2 expired* and the Closed state is entered.

The usage of timers becomes more apparent in the case when no reply is received for

Nr.	Class	Cause value	Definition
1	000	0001	Unallocated (unassigned) number
2	000	0011	No route to destination
3	001	0000	Normal/unspecified connection release
4	010	0010	Bandwidth unavailable
5	101	0001	Invalid connection reference value
6	110	0000	Mandatory information element is missing
7	110	0001	Message type non-existent or not implemented
8	110	0011	Information element/parameter non-existent or not implemented
9	110	0100	Invalid information element contents
10	110	0101	Message not compatible with call state
11	110	1000	Timer T1 expired
12	110	1001	Timer T2 expired
13	110	1111	Protocol error, unspecified
14	110	1010	TTL in SETUP expired

Table 3.14: Cause values

a RELEASE message. It is now possible for a connection to be released by a node, but if its communication with its neighbour is interrupted during the connection release, the neighbour will have resources reserved for the connection indefinitely. The addition of the T2 timer will only be able to solve this problem if we have a reliable network or a reliable transport protocol for the signalling messages. Section 3.9 shall attempt to solve this problem by introducing a reliable transport protocol.

3.7.3 Handling of error conditions

This section describes the actions to be taken when certain errors occur during connection setup.

- Whenever the integrity check of a message (using the included checksum) fails, the message shall be discarded.
- When a SETUP message is received with an illegal destination address, for example 127.0.0.1, a RELEASE CONFIRM message will be sent to the previous node with cause value *Nr. 1, "Unallocated (unassigned) number"*.
- If there is no entry in the routing table that matches the destination node, a RELEASE CONFIRM will be sent to the previous node with cause value *Nr. 2, "No route to destination"*.
- A RELEASE CONFIRM message with cause value *Nr. 4, "Bandwidth unavailable"* will be sent to the previous node if there is insufficient bandwidth available on the outgoing interface connected to the next hop.
- If a RELEASE message is received indicating a connection reference value which is not recognized as an active connection, or a connection setup in progress, a RELEASE CONFIRM message with cause value *Nr. 5, "Invalid connection reference value"* is sent in reply. Remain in the Closed state.
- If a SETUP SUCCESS message is received indicating a connection reference value which is not recognized as a connection setup in progress, a RELEASE CONFIRM message with cause value *Nr. 5, "Invalid connection reference value"* is sent in reply. Remain in the Closed state.

- When a SETUP message is received with a call reference value indicating an active connection or a connection in the process of being set up, the SETUP message should be ignored.

- When a RELEASE CONFIRM message is received with a call reference value which is not recognized as an active connection or a connection in the process of being set up, it should be ignored.

- Whenever an unexpected RELEASE message is received (for example in response to a SETUP message), the connection shall be released and a RELEASE CONFIRM message sent in reply. A RELEASE CONFIRM message is sent to the previous node in the connection and the Closed state is entered. If there is no cause value in the received RELEASE message, the cause value should be *Nr. 13, "Protocol error, unspecified"*.

- Whenever an unexpected RELEASE CONFIRM message is received (for example in the Established state), the connection shall be released. A RELEASE message should be sent to the previous node and the Release Received state entered. If there is no cause value in the received RELEASE CONFIRM message, the cause value should be *Nr. 3, "Normal/unspecified connection release"*.

- When a SETUP message is received which has one or more information elements missing, a RELEASE CONFIRM message with cause *Nr. 6, "Mandatory information element is missing"* shall be returned.

- When a RELEASE message is received with the cause value missing, the actions shall be the same as if a RELEASE message with cause value *Nr. 3, "Normal/unspecified connection release"* was received. Only, the RELEASE CONFIRM message to the node from which the RELEASE message was received shall contain a cause value *Nr. 6, "Mandatory information element is missing"*.

- When a RELEASE message is received with an invalid cause value, the actions shall be the same as if a RELEASE message with cause value *Nr. 3, "Normal/unspecified connection release"* was received. Only, the RELEASE CONFIRM message to the node from which the RELEASE message was received shall contain a cause value *Nr. 9, "Invalid information element contents"*.

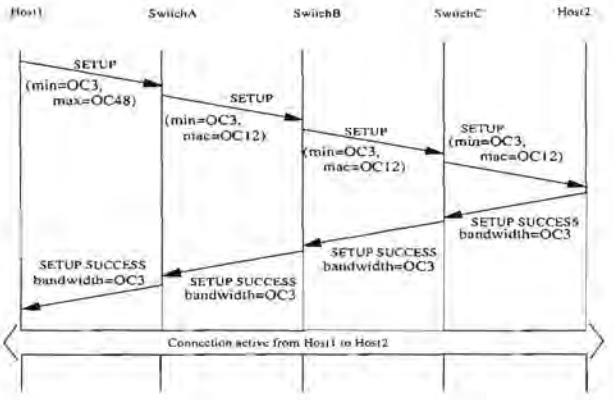


Figure 3.11: Connection setup with minimum and maximum bandwidth requirements

- When a RELEASE CONFIRM message is received with a missing or invalid cause value, it will be assumed that a RELEASE CONFIRM message with a cause value *Nr. 3, "Normal/unspecified connection release"* has been received.

3.8 Maximum bandwidth selection

When an end host requests a connection, it may specify that the minimum and maximum bandwidth should be the same, in which case all the nodes along the path shall attempt to establish a connection of the requested bandwidth. If any node cannot provide the bandwidth, it shall send the RELEASE CONFIRM message as described in Section 3.6.1.

A node can also specify a range of bandwidth, specified by a minimum and a maximum. These two values (minimum, maximum) indicate to the receiver of the SETUP message that the preceding node requests a connection of bandwidth *max bandwidth*, but if the request cannot be satisfied, the requesting host will be satisfied with any bandwidth larger than or equal to *min bandwidth*. If *max bandwidth* cannot be satisfied, but another value larger than (or equal to) *min bandwidth* can, then the SETUP message shall be changed to indicate the new maximum bandwidth - which may result in the minimum and maximum bandwidths being equal somewhere along the path. The resource reservation shall continue as specified in Section 3.6.1. This may result in different bandwidth allocations along the path of the connection, but it shall be rectified at the receipt of the SETUP SUCCESS message. The end host to which the connection is being set up shall place the last value of *max bandwidth* in the *bandwidth* information element of the SETUP SUCCESS message. While the SETUP SUCCESS message traverses the network, all nodes which form part of the connection shall edit their allocations accordingly.

For example in Figure 3.11, Host 1 requests a connection with bandwidth OC48, it also indicates that it will be satisfied by a connection with bandwidth of OC3. Switch A does not have enough resources available, but it does have resources available for an OC12 connection which it reserves for the connection. Switch A goes ahead by editing the SETUP message to reflect the changes and passes the request on to Switch B. Switch B does have enough resources available for the OC12 connection, and so has Switch C. Unfortunately, Host B only has enough resources for an OC3 connection. So, it reserves the resources for the OC3 connection and sends the SETUP SUCCESS message to Switch C with the *bandwidth* information element containing OC3. Switch C changes the allocation it made previously (only reserving resources for an OC3 connection instead of an OC12 connection) and passes the message to Switch B. In this way the message traverses the network, and an OC3 connection is successfully set up between Host A and Host B.

This solution has the unavoidable drawback that resources that are not going to be used are unavailable for a short time.

3.9 Improvements to signalling protocol

3.9.1 Routing through heterogenous nodes

So far in the discussion we assumed the network is always homogeneous. Now, this assumption is relaxed and we consider the problem associated with *heterogenous networks* (which are networks consisting of switches with different cross-connect rates). First we consider a simple example illustrated in Figure 3.12. In this example end host A requests a connection with rate OC1 to end host B. Switch S1 receives this message but its immediate neighbour has a higher cross-connect rate, and hence an OC1 connection cannot be provided.

The routing protocol which is described in Chapter 4 will construct a routing table in which the "next-hop node" entry of node I may be a node that is not directly connected through a physical or logical link to node I. It is considered a "neighbour" if it is the next-hop switch that operates at the same or lower rate as node I through which a connection must be routed.

Connection setup in a heterogenous network can now be explained as follows. If the "next-hop node" entry in the routing table indicates a node that is not an immediate neighbour of node I (physically or logically) as determined by examining the connectivity

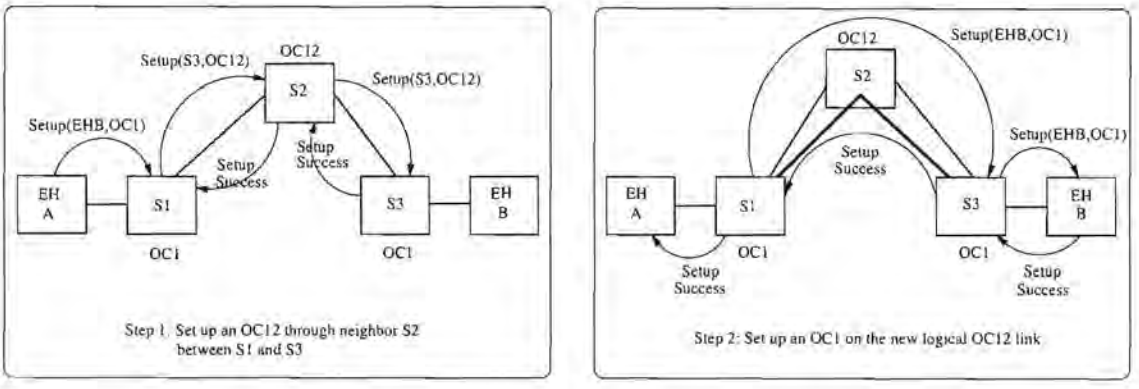


Figure 3.12: Connection setup in heterogenous network

table, the routing table is queried again to locate which immediate neighbour should be used to reach the "next-hop node" of node I. Assume that node J is the immediate neighbour and node K is the "next-hop node". Node I must determine the cross-connect rate of node J and request a connection setup at this rate between itself and node K. Once this is complete, the setting up of the lower rate connection can proceed over the newly created logical link between nodes I and K.

In the example of Figure 3.12, the routing table of switch S1 will look as follows. For data rate OC1 and destination B the "next-hop node" shall be given as S3. According to the information in the connectivity table (Table 3.5) S3 is not a neighbour of S1. So, it has to search the routing table again to look for the entry where S3 is a destination. The lowest data rate at which there is an entry for S3 shall be used as the requested rate when a connection is set up between itself and S3. Once this connection is established, nodes S1 and S3 become immediate neighbours on this newly created logical link. Switch S1 can then send the OC1 *Setup* request to switch S3 for further routing as shown in Step 2 of Figure 3.12.

Another example is given in Figure 3.13. In this example end host A starts by sending a setup request to switch S1 for a connection of rate OC12 to end host B. Switches S1, S2 and S3 can all handle the connection requirements due to their cross-connect rates being smaller or equal to the requested rate. We shall follow the progress of the connection setup from switch S3.

Switch S3 receives the setup request for an OC12 connection, to pass the connection setup request on to the next hop it queries the routing table. The routing table has an entry for data rate OC12 that indicates that switch S6 is the "next-hop node" (Table 3.15).

Destination node address	Data rate OC12			...	Data rate 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
End host B	Switch S6						
Switch S6					Switch S4		

Table 3.15: Routing table of switch 3

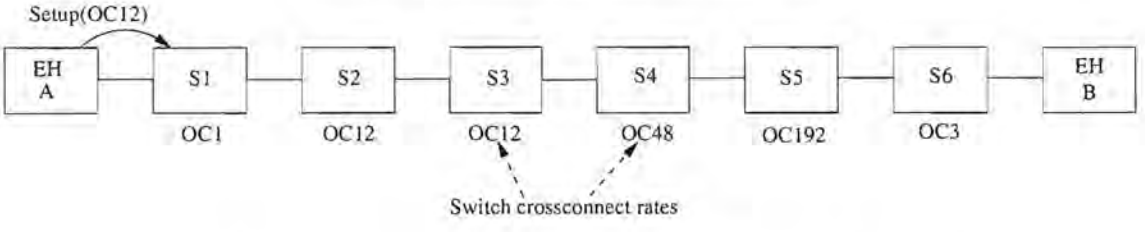


Figure 3.13: Connection setup in large heterogenous network

Querying the connectivity table, switch S3 realizes that S6 is not an immediate neighbour, so it searches the routing table again for an entry with switch S6 as a destination. Because OC192 is the highest data rate on the path to switch S6, there shall be no entry for S6 under data rate OC48, but there will be an entry with data rate OC192 which is presented in Table 3.15. According to Table 3.15 the next hop is switch S4. This switch is directly connected, so switch S3 requests a connection with rate OC192 from itself to switch S6 with the next hop being S4. After this connection is set up, S3 can send the request for an OC12 connection directly on the new logical link.

The question of when the logical link is released has not received much attention, but it should be noted that the logical link can be held open even when the connection that prompted its setup is closed. This enables the setup of other connections over this logical link without the added overhead of setting up the logical link. Considering the first example again, the logical OC12 link between switch S1 and switch S3 can be held open even if the OC1 connection closes. With the help of the routing protocol, this link can be advertised, and new OC1 connections can be set up over it. Once the last OC1 connection

closes, a timer can be started, if no new requests for OC1 connections are received before the timer's expiration, the logical link can be closed.

Note that when a connection passes through a switch with a lower crossconnect rate, for example by realizing an OC12 connection as 12 OC1 connections, some mechanism of multiplexing/demultiplexing is required. These actions have been assumed to occur transparently because of the synchronous nature of the SONET network.

3.9.2 Transport of signalling messages

Transporting the signalling messages over the IP network has already introduced two important limitations. Firstly, using IP without TCP implies unreliable transmission. The network does not provide any guarantee that the signalling message will reach its destination, there is no indication if the signalling message has reached its destination and if the message reaches the destination, there is no guarantee the data is intact. As described in Section 3.7.2, the unreliable nature of the network can cause some switches to waste valuable resources. The second limitation is security. Although this is not the focus of this research, the lack of authentication mechanisms, together with the use of a public network for the transport of the signalling messages might introduce some security risks, for example denial of service attacks, to the circuit switched network. The use of a private IP network just to transport signalling messages is wasteful of resources.

The IETF Signalling Transport working group is currently working on the Stream Control Transmission Protocol (SCTP) [22]. SCTP started out as a transport protocol for PSTN signalling messages over the IP network, but the capabilities of this protocol will solve the problems introduced when only IP is used to transfer the signalling messages. SCTP provides acknowledged error-free non-duplicated transfer of user data and is resistant to flooding and masquerade attacks [22]. However, with our goal for hardware implementation, it still needs to be determined if SCTP or TCP can be implemented in hardware.

Another option is to carry the signalling messages using SONET. Inside a SONET frame there are three section overhead bytes (named D1, D2 and D3) and eight line overhead bytes (named D4 to D12). These bytes are termed the section data communications channel (DCC) and line data communication channel (DCC) respectively and are used for operations, administration, maintenance and provisioning (OAM&P). Carrying signalling messages on one of these point-to-point DCC channels will eliminate the need for an IP

network.

3.10 Conclusion

Signalling protocols in current TDM networks only provide low-bandwidth (DS0) circuits on-demand to one application (telephony). This chapter describes a new hardware implementable signalling protocol that is able to provide high-bandwidth on-demand circuits for applications such as bulk data transfers. The signalling protocol presented in this chapter requires a supporting routing protocol. The routing protocol that will be presented in Chapter 4 is responsible for the creation of the routing table entries that are required by the signalling protocol.