Computer Communication Networks — Lecture Notes

D.B. Hoang and K.J. Pye

1995 edition
(7.54 p.m. 31 January, 1996)

School of Electronic Engineering, La Trobe University

Preface

These notes are very heavily based in parts on notes originally written by Dr D.B. Hoang. They have since been modified and extended by Mr K.J. Pye. Consequently all mistakes are the responsibility of Mr Pye who would be grateful to receive comments on how these notes might be improved.

Contents

Chapter 1. Network Architecture

1.1. Introduction

Computer networks are designed in a highly structured way to reduce their design complexity. Most networks are organised as a series of *layers* or levels. The number of layers, the name of each layer, and the function of each layer differs from network to network. However, in all networks, each layer clearly defines various data communication functions and logical operations. Each level is functionally independent of the others, but builds on its predecessor. In order to function, higher levels depend on correct operation of the lower levels.

Figure 1.1 illustrates a 7-layer network architecture. Layer (level) *n* on one computer carries on communication with layer *n* on another computer. The set of rules and conventions that encompasses electrical, mechanical and functional characteristics of a data link, as well as the control procedures for such communication is called the *layer n protocol.*

The communication between two layers at the same level (layer *n*, $n \neq 1$) of two different computers is called *virtual communication.* Here, each layer passes data and control information to the layer immediately below it, until the lowest layer (layer 1). At layer 1, information from one computer is physically transferred to layer 1 of the other *(physical communication).*

The *interface* between each pair of adjacent layers defines which operations and services the lower layer offers to the upper one.

The *network architecture* thus can be defined as the set of layers and protocols.

1.2. The ISO and other models

Figure 1.2 shows the reference model of the Open Systems Interconnection (OSI), which has been developed by the International Standards Organisation (ISO). We will briefly define the functions and operation of each layer of this architecture in turn.

1.2.1. Layer 1: the physical layer

This layer is concerned with transmitting an electrical signal representation of data over a communication link. Typical conventions would be: voltage levels used to represent a “1” and a “0”, duration of each bit, transmission rate, mode of transmission, and functions of pins in a connector. An example of a physical layer protocol is the RS-232 standard.

1.2.2. Layer 2: the data link layer

This layer is concerned with error-free transmission of data units. The data unit is an abbreviation of the official name of *data-link-service-data-units;* it is sometimes called the *data frame.* The function of the data link layer is to break the input data stream into data frames, transmit the

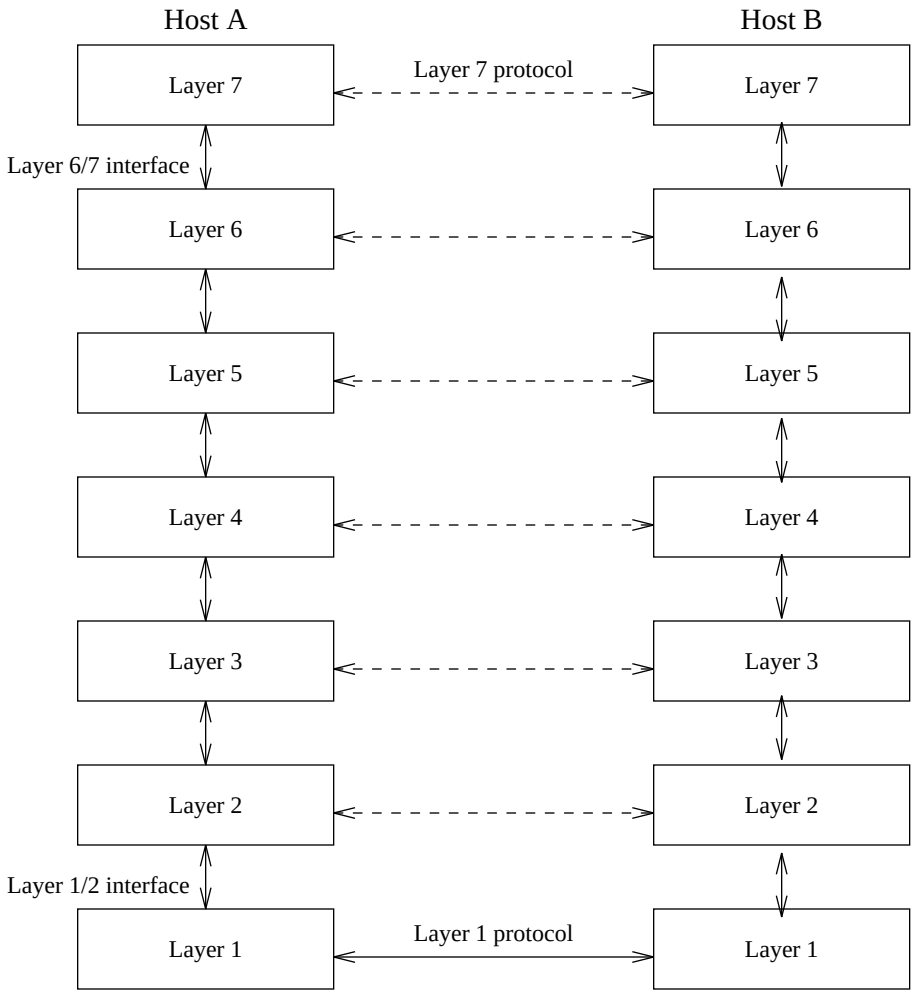


Figure 1.1. Layers, protocols and interfaces

frames sequentially, and process the *acknowledgement frame* sent back by the receiver. Data frames from this level when transferred to layer 3 are assumed to be error free.

1.2.3. Layer 3: the network layer

This layer is the *network control* layer, and is sometimes called the *communication subnet layer*. It is concerned with intra-network operation such as addressing and routing within the subnet.

Basically, messages from the source host are converted to *packets*. The packets are then routed to their proper destinations.

1.2.4. Layer 4: the transport layer

This layer is a *transport end-to-end control layer* (i.e. source-to-destination). A program on the source computer communicates with a similar program on the destination computer using the message headers and control messages, whereas all the lower layers are only concerned with

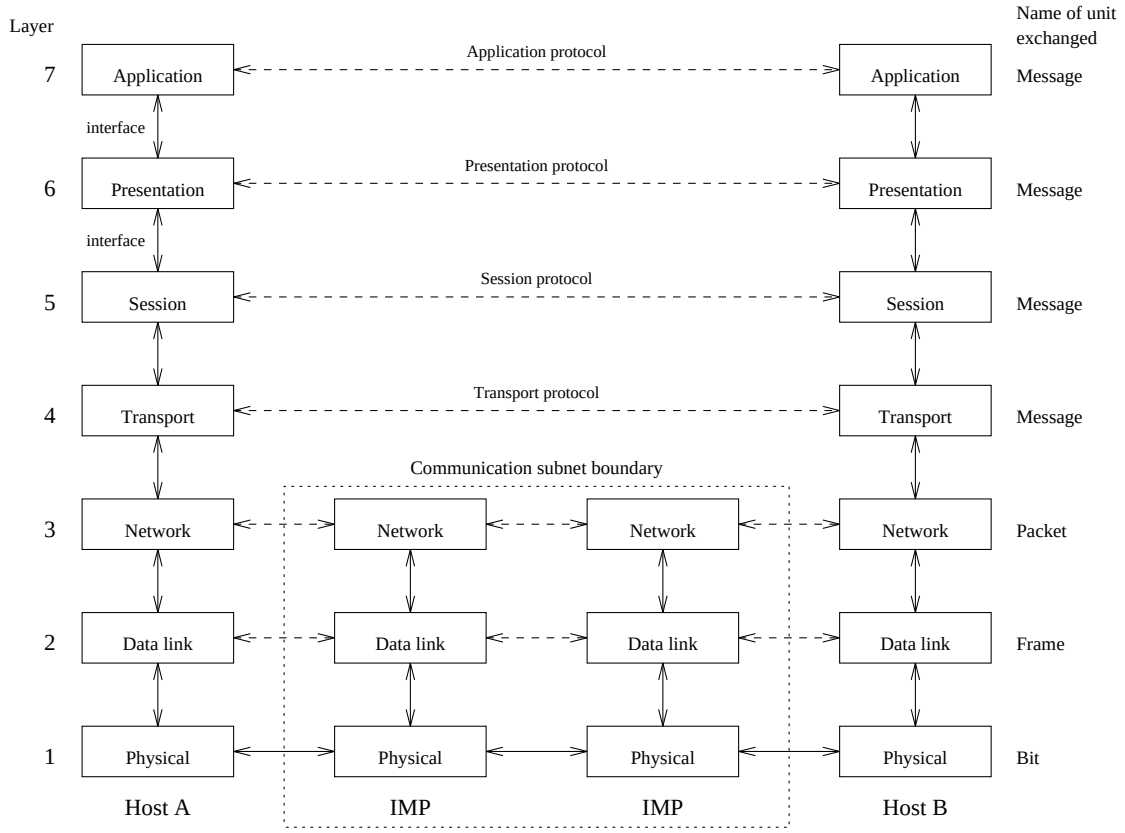


Figure 1.2. The OSI model

communication between a computer and its immediate neighbours, not the ultimate source and destination computers.

The transport layer is often implemented as part of the operating system. The data link and physical layers are normally implemented in hardware.

1.2.5. Layer 5: the session layer

The session layer is the user's interface into the network. This layer supports the dialogue through session control, if services can be allocated. A connection between users is usually called a *session.* A session might be used to allow a user to log into a system or to transfer files between two computers. A session can only be established if the user provides the remote addresses to be connected. The difference between session addresses and transport addresses is that session addresses are intended for users and their programs, whereas transport addresses are intended for transport stations.

1.2.6. Layer 6: the presentation layer

This layer is concerned with transformation of transferred information. The controls include message compression, encryption, peripheral device coding and formatting.

1.2.7. Layer 7: the application layer

This layer is concerned with the application and system activities. The content of the application layer is up to the individual user.

1.3. Examples of Network Architectures

1.3.1. IBM's System Network Architecture (SNA)

IBM's System network architecture is a method for unifying network operations. SNA describes the division of network functions into discrete layers and defines protocols and formats for communication between equivalent layers. SNA describes a network in terms of a physical network and a logical network. The physical network consists of a collection of *nodes*: host node, front end communication node, concentration node and terminal node. The host node is the central processor; the front end communication node is concerned with data transmission functions; the concentration node supervises the behaviour of terminals and other peripherals; the terminal node is concerned with the input and output of information through terminal devices. Figure 1.3 depicts a simple SNA network.

The SNA logical network consists of three layers:

(i) transmission management;

(ii) function management; and

(iii) application.

Each node in the SNA physical network may contain any or all of these three layers. Communication between layers is as shown in Figure 1.4 below.

The application layer consists of the user's application programs and is concerned only with the processing of information.

The functional management layers controls the presentation format of information sent from and received by the application layer, i.e. it converts th data into a form convenient to the user.

The transmission management layer controls movement of user data through the network. It involves routing, scheduling and transmission functions. This layer exists in every intermediate node through which the data units flow and may utilise a variety of physical connections and protocols between the nodes of an SNA network.

Each node contains one or more *Network Addressable Units* (NAU). There are three types of NAU. A *Logical Unit* (LU) is a NAU which users use to address their process. A *Physical Unit* (PU) is a NAU which the network uses to address a physical device, without reference to which processes are using it. The third type of NAU is the *System Services Control Point* (SSCP) which has control over all front ends, remote concentrators and terminals attached to the host.

The three types of NAU communicate with each other by invoking the services of the transmission management layer as shown in Figure 1.5.

The *physical link control* protocol takes care of electrical transmission of data bits from one node

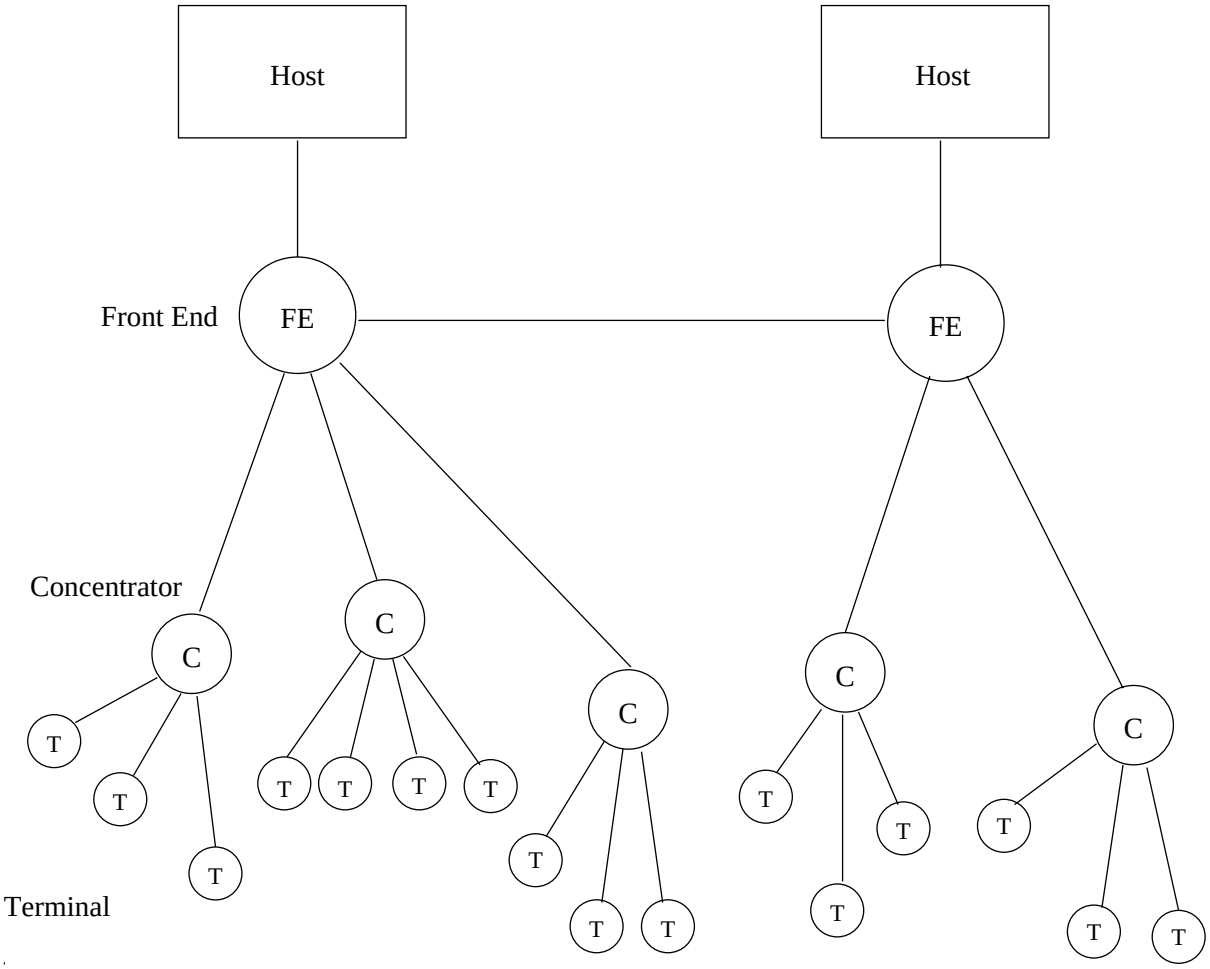


Figure 1.3.

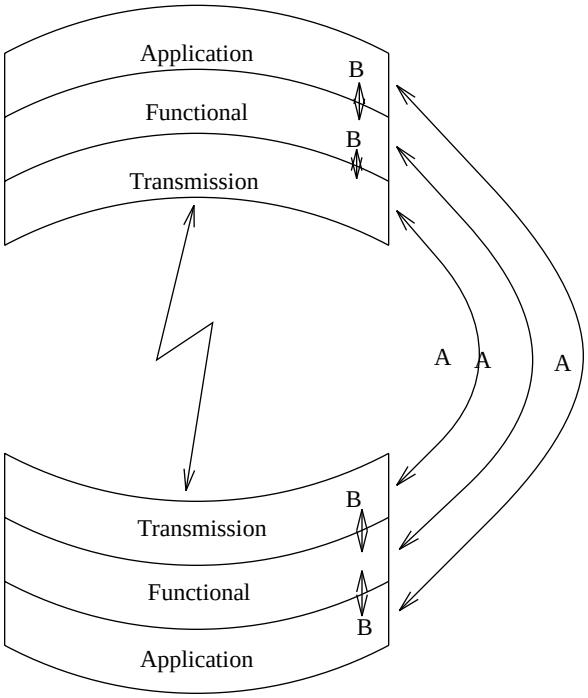
to another. The *data link control* protocol constructs frames from the data stream, detecting and recovering from transmission errors. This level 2 protocol is called SDLC (Synchronous Data Link Control) which we will look at later. The *path control* protocol performs the path-selection and congestion control functions within the subnet. The *transmission control* protocol initiates, recovers and terminates transport connections (called sessions) in SNA. It also controls the flow of data to and from other NAUs.

The SNA and ISO models do not correspond closely. However, a rough comparison of the architecture control levels is shown in Figure 1.6.

1.3.2. DECNET's DNA (Digital Network Architecture)

A DECNET is just a collection of computers (nodes) whose functions include running user programs, performing packet switching or both. The architecture of DECNET is called Digital Network Architecture (DNA).

DNA has five layers. The physical layer, data link control layer, transport layer and network services layer correspond to the lowest four OSI layers. DNA does not have a session layer (layer



A: Peer layer communication

B: Adjacent layer communication

Figure 1.4.

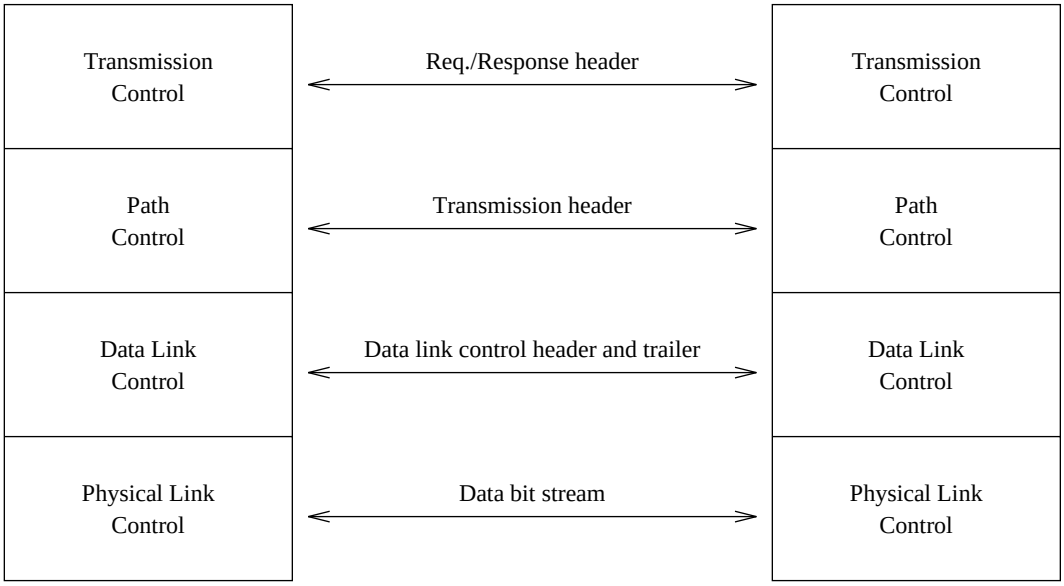


Figure 1.5.

5) and its application layer corresponds to a mixture of the presentation and application layers in OSI; see Figure 1.6. Notice that DNA's level 3 is called the transport layer, not level 4 as in the

<table>
<tr><th>Layer</th><th>OSI</th><th>SNA</th><th>DECNET</th></tr>
<tr><td>7</td><td>Application</td><td>End User</td><td rowspan="2">Application</td></tr>
<tr><td>6</td><td>Presentation</td><td>NAU services</td></tr>
<tr><td>5</td><td>Session</td><td>Data Flow Control</td><td>(none)</td></tr>
<tr><td>4</td><td>Transport</td><td>Transmission Control</td><td>Network Service</td></tr>
<tr><td>3</td><td>Network</td><td rowspan="2">Path Control</td><td>Transport</td></tr>
<tr><td>2</td><td>Data Link</td><td>Data Link</td><td>Data Link</td></tr>
<tr><td>1</td><td>Physical</td><td>Physical</td><td>Physical</td></tr>
</table>

Figure 1.6. Comparison of architecture control levels

OSI model.

Chapter 2. The Data Link Layer

2.1. Introduction

In this section we will study the design of layer 2, the data link layer (also known as the physical link control layer).

The purpose of the data link layer is to transfer blocks of data without error between two adjacent devices. Adjacent devices are physically connected by a communication channel such as telephone lines, coaxial cables, optical fibres, or satellites. The implication of such a physical link is that the data bits are delivered in exactly the same order in which they are sent. The physical link has no inherent storage capacity, therefore the delay involved is the propagation delay over the link.

Transmission of data over the link would be very simple indeed if no error ever occurred. Unfortunately, this is not so in a real physical link for a number of reasons:

- Natural phenomena such as noises and interference are introduced into the link causing errors in detecting the data.
- There is a propagation delay in the link.
- There is a finite data processing time required by the transmitting and receiving stations.

A data link protocol thus has to be designed to ensure an error-free transmission and also to achieve an efficiency of the data transfer as high as possible.

2.2. Simplified Model

At layer 2, user's messages are already broken up into segments. Control information (headers and trailers) is added to each segment to make up a frame. We are concerned only with the transmission of frames between two adjacent IMPs. For this reason the simplified model of Figure 2.2 will be sufficient for our study.

2.3. Functions and requirements of the Data Link Protocols

The basic function of the layer is to transmit frames over a physical communication link. Transmission may be *half duplex* or *full duplex*. To ensure that frames are delivered free of errors to the destination station (IMP) a number of requirements are placed on a data link protocol. The protocol (control mechanism) should be capable of performing:

1. The identification of a frame (i.e. recognise the first and last bits of a frame).

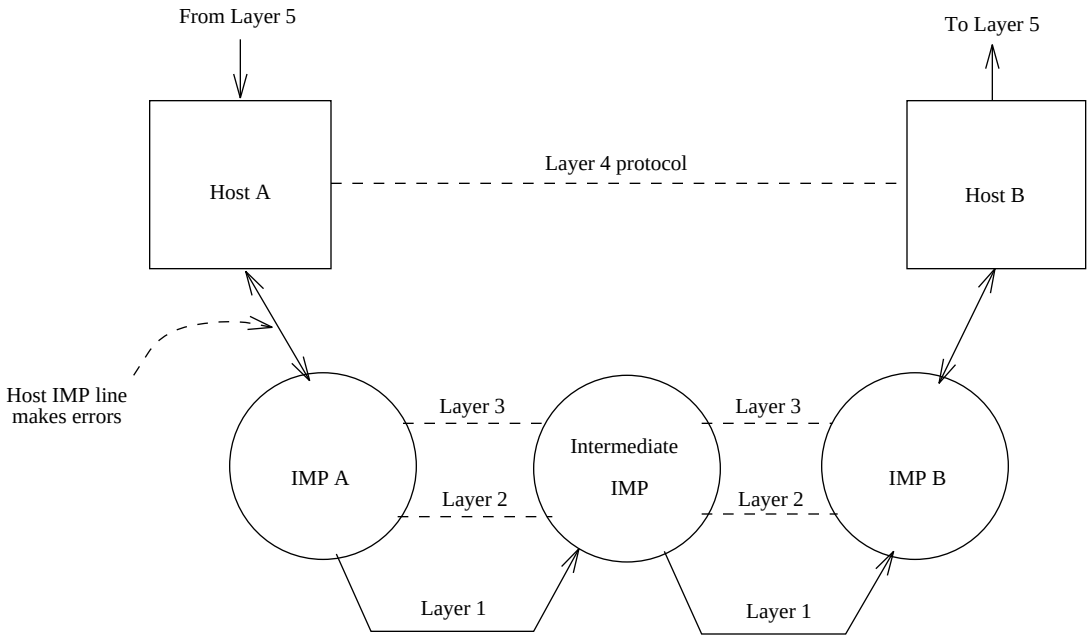


Figure 2.1. Layer 2, 3, and 4 protocols in the general case

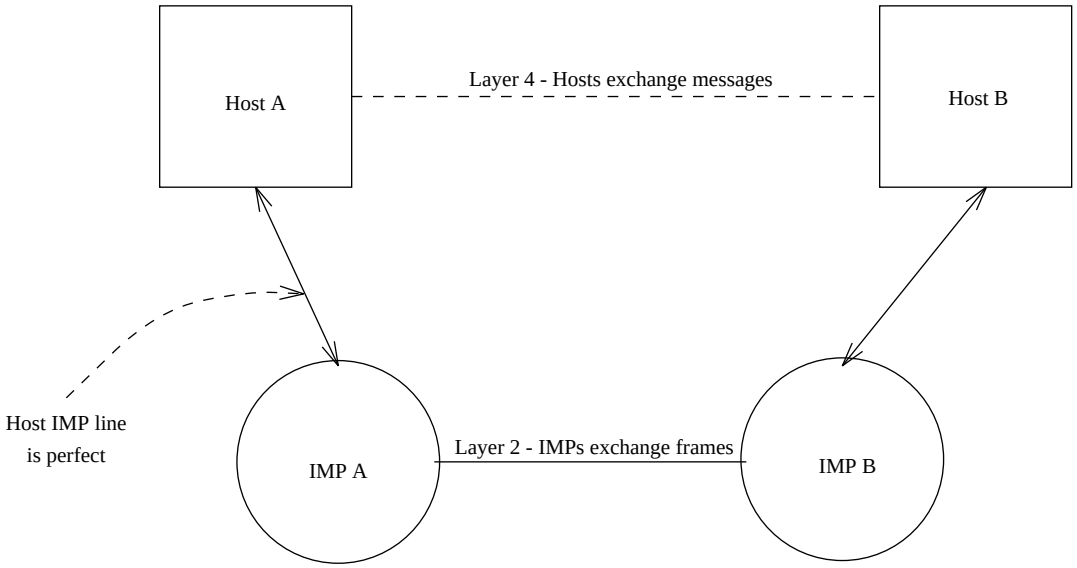


Figure 2.2. Simplified model used in this section

2. The transmission of frames of any length up to a given maximum. Any bit pattern is permitted in a frame.

3. The detection of transmission errors.

4. The retransmission of frames which were damaged by errors.

5. The assurance that no frames were lost.

6. In a multidrop configuration

(i) Some addressing mechanism must be used to direct the frames.

(ii) Some mechanism must be used for preventing conflicts caused by simultaneous transmission by many stations.

7. The detection of failure or abnormal situations for control and monitoring purposes.

It should be noted that as far as layer 2 is concerned a host message is pure data, every single bit of which is to be delivered to the other host. The frame header pertains to layer 2 and is never given to the host.

We will first look at three elementary data link protocols of increasing complexity.

2.4. Elementary Data Link Protocols

2.4.1. An unrestricted simplex protocol

In order to appreciate the step by step development of efficient and complex protocols such as SDLC, HDLC etc., we will begin with a simple but unrealistic protocol. In this protocol:

- Data are transmitted in one direction only
- The transmitting (Tx) and receiving (Rx) hosts are always ready
- Processing time can be ignored
- Infinite buffer space is available
- No errors occur; i.e. no damaged frames and no lost frames.

The protocol consists of two procedures, a sender and receiver as depicted below:

```
/* protocol 1 */

Sender()
{
      forever
      {
            from_host(buffer);
            S.info = buffer;
            sendf(S);
      }
}

Receiver()
{
      forever
```

```
    {
        wait(event);
        getf(R);
        to_host(R.info);
    }
}
```

2.4.2. A simplex stop-and-wait protocol

In this protocol we assume that

- Data are transmitted in one direction only
- No errors occur
- The receiver can only process the received information at a finite rate

These assumptions imply that the transmitter cannot send frames at a rate faster than the receiver can process them.

The problem here is how to prevent the sender from flooding the receiver.

A general solution to this problem is to have the receiver provide some sort of feedback to the sender. The process could be as follows: The receiver send an acknowledge frame back to the sender telling the sender that the last received frame has been processed and passed to the host; permission to send the next frame is granted. The sender, after having sent a frame, must wait for the acknowledge frame from the receiver before sending another frame. This protocol is known as *stop-and-wait*.

The protocol is as follows:

```
/* protocol 2 */

Sender()
{
    forever
    {
        from_host(buffer);
        S.info = buffer;
        sendf(S);
        wait(event);
    }
}

Receiver()
{
    forever
    {
        wait(event);
```

```
            getf(R);
            to_host(R.info);
            sendf(S);
        }
}
```

2.4.3. A simplex protocol for a noisy channel

In this protocol the unreal "error free" assumption in protocol 2 is dropped. Frames may be either damaged or lost completely. We assume that transmission errors in the frame are detected by the hardware checksum.

One suggestion is that the sender would send a frame, the receiver would send an ACK frame only if the frame is received correctly. If the frame is in error the receiver simply ignores it; the transmitter would time out and would retransmit it.

One fatal flaw with the above scheme is that if the ACK frame is lost or damaged, duplicate frames are accepted at the receiver without the receiver knowing it.

Imagine a situation where the receiver has just sent an ACK frame back to the sender saying that it correctly received and already passed a frame to its host. However, the ACK frame gets lost completely, the sender times out and retransmits the frame. There is no way for the receiver to tell whether this frame is a retransmitted frame or a new frame, so the receiver accepts this duplicate happily and transfers it to the host. The protocol thus fails in this aspect.

To overcome this problem it is required that the receiver be able to distinguish a frame that it is seeing for the first time from a retransmission. One way to achieve this is to have the sender put a sequence number in the header of each frame it sends. The receiver then can check the sequence number of each arriving frame to see if it is a new frame or a duplicate to be discarded.

The receiver needs to distinguish only 2 possibilities: a new frame or a duplicate; a 1-bit sequence number is sufficient. At any instant the receiver expects a particular sequence number. Any wrong sequence numbered frame arriving at the receiver is rejected as a duplicate. A correctly numbered frame arriving at the receiver is accepted, passed to the host, and the expected sequence number is incremented by 1 (modulo 2).

The protocol is depicted below:

```
/* protocol 3 */

Sender()
{
    NFTS = 0;           /* NFTS = Next Frame To Send */
    from_host(buffer);
    forever
    {
        S.seq = NFTS;
        S.info = buffer;
        sendf(S);
        start_timer(S.seq);
        wait(event);
```

```
            if(event == frame_arrival)
            {
                from_host(buffer);
                ++NFTS;
            }
        }
}

Receiver()
{
        FE = 0;                 /* FE = Frame Expected */
        forever
        {
            wait(event);
            if(event == frame_arrival)
            {
                getf(R);
                if(R.seq == FE)
                {
                    to_host(R.info);
                    ++FE;
                }
                sendf(S);       /* ACK */
            }
        }
}
```

This protocol can handle lost frames by timing out. The timeout interval has to be long enough to prevent premature timeouts which could cause a “deadlock” situation.

2.5. Sliding Window Protocols

2.5.1. Piggybacking technique

In most practical situations there is a need for transmitting data in both directions (i.e. between 2 computers). A full duplex circuit is required for the operation.

If protocol 2 or 3 is used in these situations the data frames and ACK (control) frames in the reverse direction have to be interleaved. This method is acceptable but not efficient. An efficient method is to absorb the ACK frame into the header of the data frame going in the same direction. This technique is known as *piggybacking.*

When a data frame arrives at an IMP (receiver or station), instead of immediately sending a separate ACK frame, the IMP restrains itself and waits until the host passes it the next message. The acknowledgement is then attached to the outgoing data frame using the ACK field in the frame header. In effect, the acknowledgement gets a free ride in the next outgoing data frame.

This technique makes better use of the channel bandwidth. The ACK field costs only a few bits,

whereas a separate frame would need a header, the acknowledgement, and a checksum.

An issue arising here is the time period that the IMP waits for a message onto which to piggyback the ACK. Obviously the IMP cannot wait forever and there is no way to tell exactly when the next message is available. For these reasons the waiting period is usually a fixed period. If a new host packet arrives quickly the acknowledgement is piggybacked onto it; otherwise, the IMP just sends a separate ACK frame.

2.5.2. Sliding window

When one host sends traffic to another it is desirable that the traffic should arrive in the same *sequence* as that in which it is dispatched. It is also desirable that a data link should deliver frames in the order sent.

A flexible concept of sequencing is referred to as the *sliding window* concept and the next three protocols are all sliding window protocols.

In all sliding window protocols, each outgoing frame contains a sequence number N_S ranging from 0 to $2^n - 1$ (where n is the number of bits reserved for the sequence number field).

At any instant of time the sender maintains a list of consecutive sequence numbers corresponding to frames it is permitted to send. These frames are said to fall within the *sending window*. Similarly, the receiver maintains a *receiving window* corresponding to frames it is permitted to accept.

Figure 2.3 illustrates the window concept.

The size of the window relates to the available buffers of a receiving or sending node at which frames may be arranged into sequence.

At the receiving node, any frame falling outside the window is discarded. Frames falling within the receiving window are accepted and arranged into sequence. Once sequenced, the frames at the left of the window are delivered to the host and an acknowledgement of the delivered frames is transmitted to their sender. The window is then rotated to the position where the left edge corresponds to the next expected frame, N_r.

Whenever a new frame arrives from the host, it is given the next highest sequence number, and the upper edge of the sending window is advanced by one. The sequence numbers within the sender's window represent frames sent but as yet not acknowledged. When an acknowledgement comes in, it gives the position of the receiving left window edge which indicates what frame the receiver expects to receive next. The sender then rotates its window to this position, thus making buffers available for continuous transmission.

Figure 2.4 shows an example with a maximum window size of 1.

2.5.3. A one bit sliding window protocol: protocol 4

The sliding window protocol with a maximum window size 1 uses stop-and-wait since the sender transmits a frame and waits for its acknowledgement before sending the next one.

```
/* protocol 4 */

Send_and_receive()
{
```

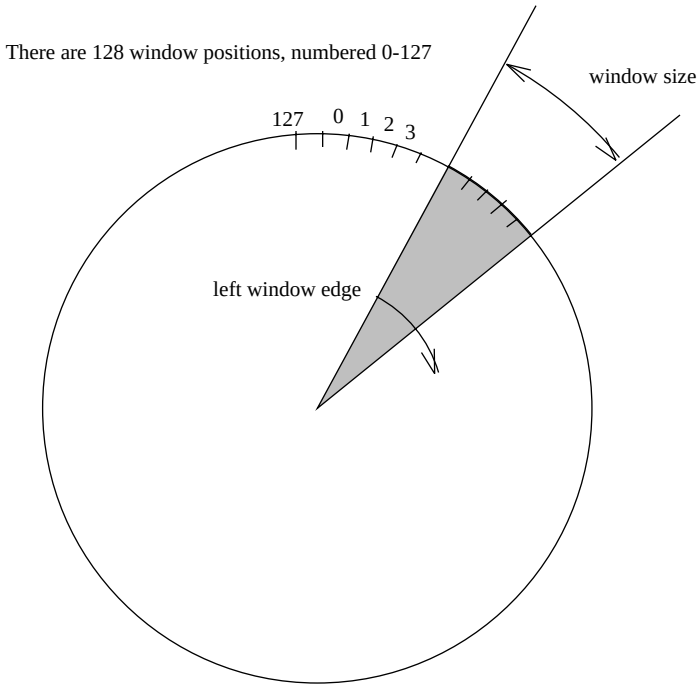


Figure 2.3. 128-position counter with a window size of 5

```
NFTS = 0;
FE = 0;
from_host(buffer);
S.info = buffer;
S.seq = NFTS;
S.ack = 1-FE;
sendf(S);
start_timer(S.seq);
forever
{
      wait(event);
      if(event == frame_arrival)
      {
            getf(R);
            if(R.seq == FE)
            {
                  to_host(R.info);
                  ++FE;
            }
            if(R.ack == NFTS)
            {
                  from_host(buffer);
                  ++NFTS;
            }
```

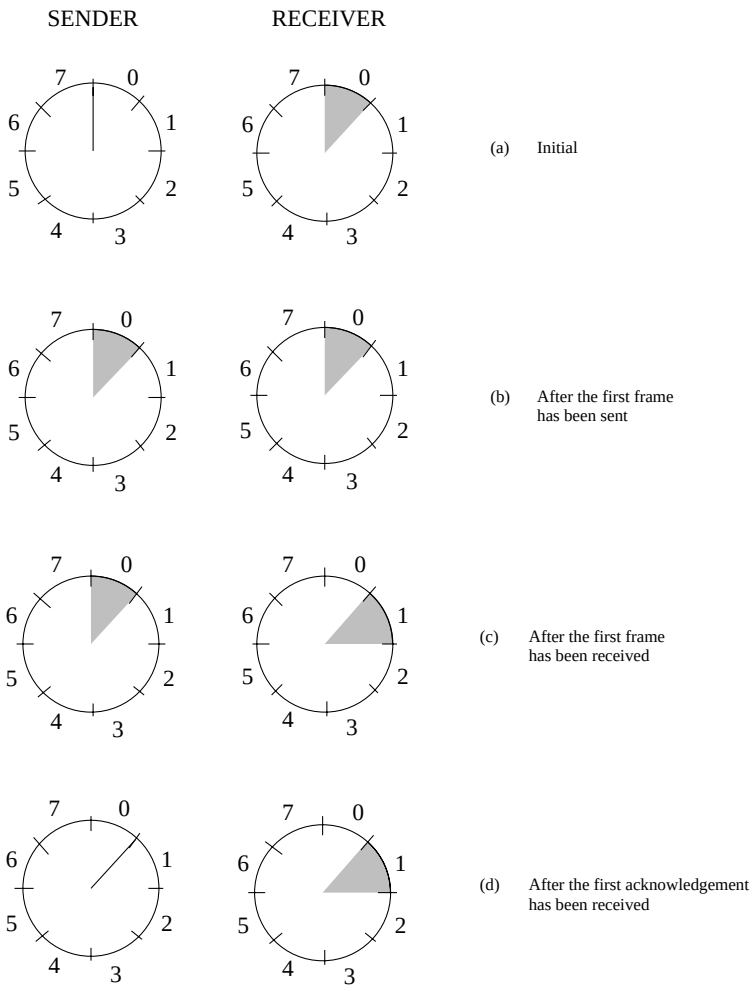


Figure 2.4. A sliding window of size 1, with a 3-bit sequence number

```
        }
        S.info = buffer;
        S.seq = NFTS;
        S.ack = 1-FE;
        sendf(S);
        start_timer(S.seq);
    }
}
```

2.5.4. Pipelining

In many situations the long round-trip time can have important implications for the efficiency of the bandwidth utilisation.

As an example, consider a satellite channel with a 500ms round-trip propagation delay. At time $t = 0$ the sender starts sending the first frame. Not until at least $t \geq 500$ ms has the acknowledgement arrived back at the sender. This means that the sender was blocked most of the time causing a reduction in efficiency.

As another example, if the link is operated in a two-way alternating mode (half-duplex), the line might have to be "turned around" for each frame in order to receive an acknowledgement. This acknowledgement delay could severely impact the effective data transfer rate.

The effects of these problems can be overcome by allowing the sender to transmit multiple contiguous frames (say up to *w* frames) before it receives an acknowledgement. This technique is known as *pipelining.*

In the satellite example, with a channel capacity of 50kbps and 1000-bit frames, by the time the sender has finished sending 26 frames, $t = 520$ ms, the acknowledgement for frame 0 will have just arrived, allowing the sender to continue sending frames. At all times, 25 or 26 unacknowledged frames will be outstanding, and the sender's window size needs to be at least 26.

Pipelining frames over an unreliable communication channel raises some serious issues. What happens if a frame in the middle of a long stream is damaged or lost? What should the receiver do with all the correct frames following the bad one?

The are two basic *Automatic Request for Repeat (ARQ)* methods for dealing with errors in the presence of pipelining.

One method, the normal mode of ARQ is called *Go-back-N*. If the receiver detects any error in frame *N*, it signals the sender and then discards any subsequent frame (Figure 2.5). The sender, which may currently be sending frame $N + X$ when the error signal is detected, initiates retransmission of frame *N* and all subsequent frames (Figure 2.6).

The other method is called *selective reject*. In this method the receiver stores all the correct frames following the bad one. When the sender finally notices what was wrong, it just retransmits the one bad frame, not all its successors. This case is shown in Figure 2.7.

2.5.5. Protocol 5: Pipelining, Multiple outstanding frames

In this protocol, the sender may transmit up to *MaxSeq* frames without waiting for an acknowledgement. In addition, unlike the previous protocols, the host is not assumed to have a new message all the time. Instead, the host causes *host ready* events when there is a message to send.

This protocol employs the Go-back-N technique. In the example below, the window size of the receiver is equal to 1, and a maximum of *MaxSeq* frames may be outstanding at any instant.

```
/* protocol 5 */

send_data(frame_number)
{
      S.info = buffer[frame_number];
      S.seq = frame_number;
      S.ack = (FE+MaxSeq) % (MaxSeq+1);
      sendf(S);
      start_timer(frame_number);
}
send_receive()
{
```

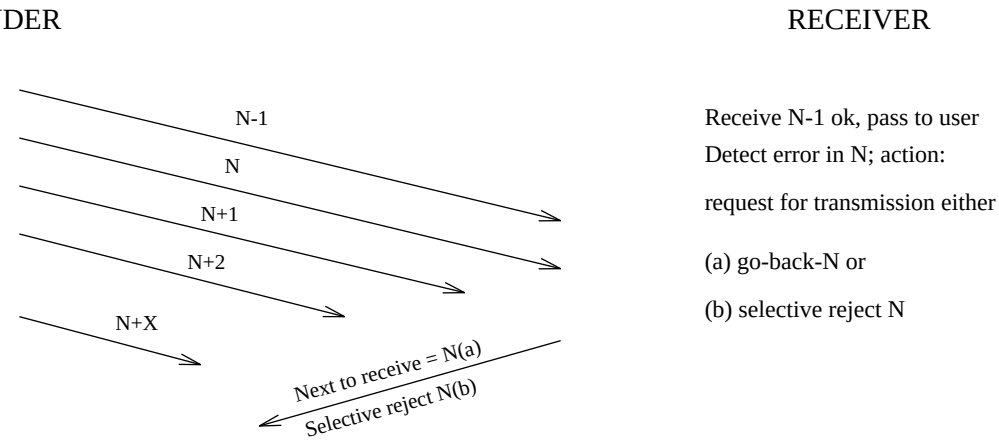


Figure 2.5. Transmit sequence

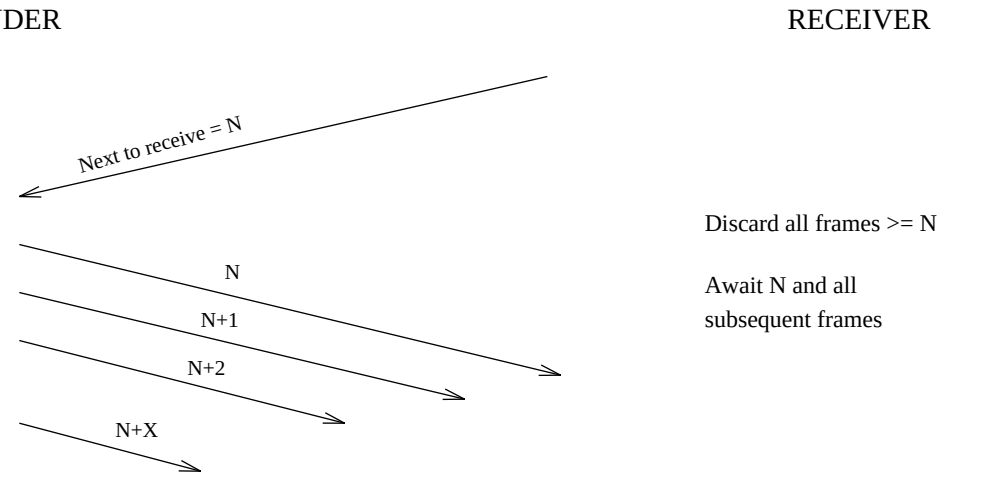


Figure 2.6. Go-back-N

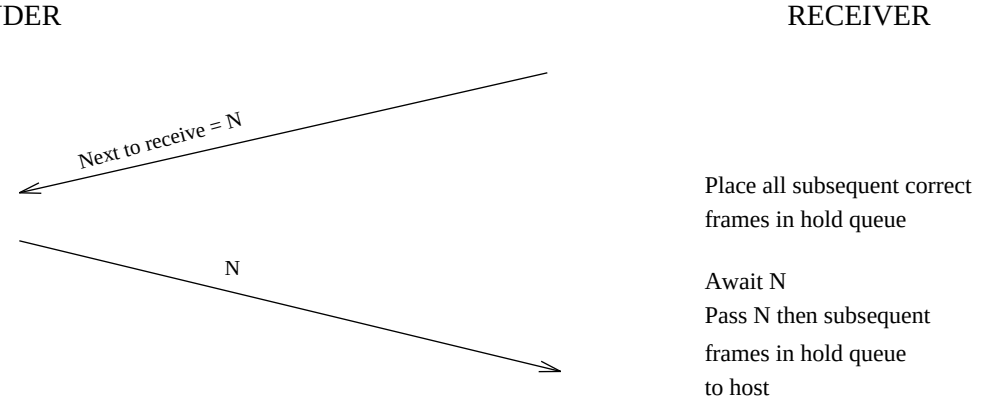


Figure 2.7. Selective reject N

```
enable_host();
NFTS = 0;
```

```
Ack_expected = 0;
Frame_expected = 0;
nbuffered = 0;
forever
{
    wait(event);
    switch(event)
    {
    case host_ready:
        from_host(buffer[NFTS]);
        ++nbuffered;
        send_data(NFTS);
        ++NFTS;
        break;
    case frame_arrival:
        getf(R);
        if(R.seq == Frame_expected)
        {
            to_host(R.info);
            ++Frame_expected;
        }
        if( (Ack_expected <= R.ack && R.ack < NFTS)
          ||(NFTS < Ack_expected && Ack_expected <= R.ack)
          ||(R.ack < NFTS &&NFTS < Ack_expected))
        {
            –nbuffered;
            stop_timer(Ack_expected);
            ++Ack_expected;
        }
        break;
    case checksum_error:
        /* just ignore the bad frame */
        break;
    case timeout:
        NFTS = Ack_expected;
        i = 0;
        do
        {
            send_data(NFTS);
            ++NFTS;
            ++i;
        } while(i<=nbuffered);
        break;
    }
    if(nbuffered < MaxSeq)
        enable_host();
```

```
            else
                disable_host();
        }
}
```

2.5.6. Protocol 6

This protocol employs the selective reject technique. The protocol does not discard good frames because an earlier frame was damaged or lost provided that these good frames fall within the receiving window.

Associated with each outstanding frame is a timer. When the timer goes off, (or when the transmitter is notified of any error), only that one frame is retransmitted, not all the outstanding frames, as in protocol 5.

In this protocol, the receiver's window size is fixed, the maximum of which is

$$(\frac{MaxSeq + 1}{2}).$$

The maximum number is thus chosen to ensure that there will not be an overlapping between the new window and the previous window. The overlapping of windows means that the receiver would not be able to differentiate between a new frame and a retransmitted frame.

The receiver has a buffer reserved for each sequence number within its window. Whenever a frame arrives, its sequence number is checked to see if it falls within the receiver's window. If so, and if it has not already been received, it is accepted and stored regardless of whether or not it is the next frame expected by the host. Frames passed to the host must always be in order.

Protocol 6 is more efficient than protocol 5 in that:

- Protocol 6 employs an auxiliary timer to prevent delay in piggybacking. If no reverse traffic has presented itself before the timer goes off, a separate acknowledgement is sent.
- Whenever the receiver has reason to suspect that an error has occurred it sends a negative acknowledgement (NAK) frame back to the sender. Such a frame is a request for retransmission of the frame specified in the NAK.

2.6. High-level Data Link Control Procedures: HDLC, SDLC

2.6.1. Introduction

In this subsection we describe the International Standards Organisation data link protocol HDLC (High-level Data Link Control). CCITT Recommendation X.25 level 2 is one of the permissible options of HDLC. All these bit-oriented protocols grew out from the original IBM SDLC (Synchronous Data Link Control). All these protocols are based on the same protocols. They differ only in minor ways (see the appropriate protocol definition).

HDLC is a discipline for the management of information transfer over a data communication channel.

HDLC has a basic structure that governs the function and the use of control procedures. The basic structure includes:

- The definitions of primary and secondary station responsibilities.
- The design of information grouping for control and checking.
- The design of the format for transfer of information and control data.

2.6.2. Primary and secondary stations

A data link involves two or more participating stations. For control purposes one station on the link is designated a *primary* station, the others are *secondary* stations. The primary station is responsible for the organisation of data flow and for the link level error recovery procedures.

A frame sent from a primary station is referred to as a *command* frame. A frame from a secondary to a primary is referred to as a *response* frame. Normally when a command is sent, a response or a string of responses is expected in reply (see Figure 2.8).

On a point-to-point link either station could be the primary. On multidrop links and where polling is employed, the station which polls the other is the primary. A station may have several links connected to it. In some configurations it may be assigned as a primary for some links and a secondary for others (Figure 2.9). The station assignments are normally made when a system is designed.

2.6.3. Frame structure

In HDLC the input data string to be transmitted over the link is broken up into *data frames* which are then transmitted sequentially. The input data string could be command, response or information. Each frame conforms to the format of Figure 2.10.

All frames start and end with the flag byte 01111110. There is a 3-byte header at the start of the frame and a 3-byte trailer at the end. Between the header and the trailer any number of bits can be carried (bit-oriented).

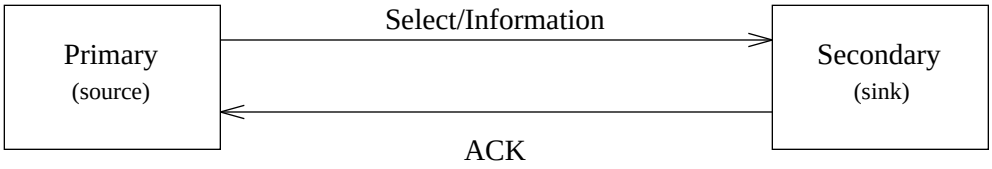
The header

byte 1 - Flag (01111110):

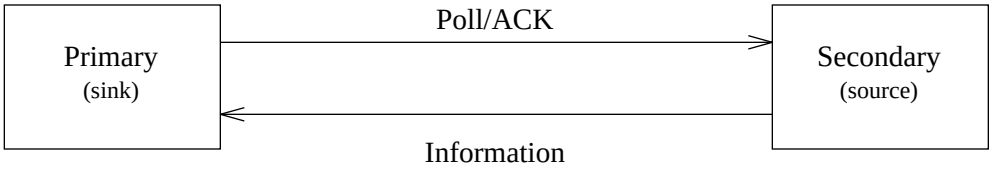
> A frame is identified by this beginning flag F and contains only non-F bit patterns. HDLC uses bit-stuffing to achieve data transparency. Whenever the transmitting hardware encounters five consecutive ones in the data, it automatically stuffs a zero bit into the outgoing bit stream. When the receiver sees five consecutive incoming one bits followed by a zero bit, it automatically deletes the zero bit. If the user data contained the flag pattern 01111110, it would be transmitted as 011111010 but stored in the receiver's memory as 01111110. Thus the pattern F can never occur by chance.

byte 2 - Address:

> In command frames, the address identifies the station(s) for which the command is intended.



(a)



(b)

Figure 2.8.

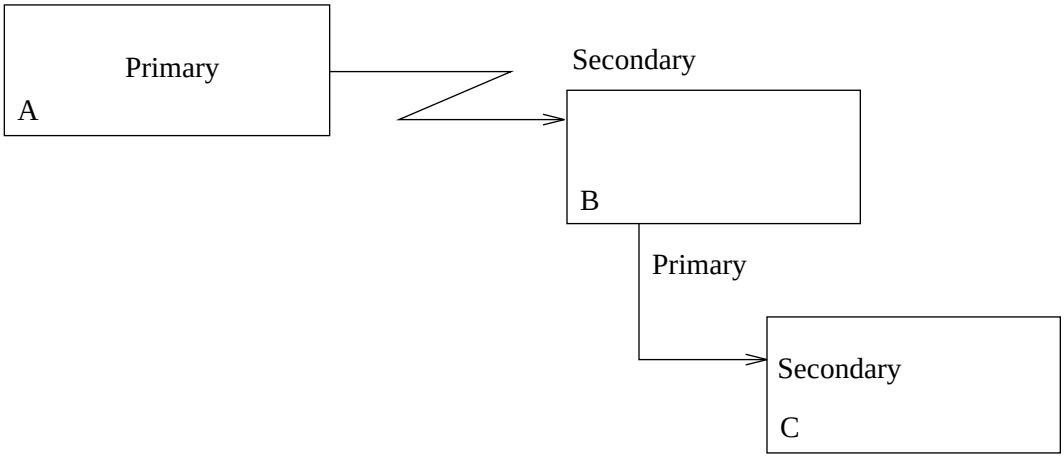


Figure 2.9.

In response frames, the address identifies the station from which the response originated. However, the address could also be a *broadcast* address so that a single frame is received by all stations on the link, or it could be the address of a group of stations.

byte 3 - Control:

The control field contains information for controlling the link, which we will examine later.

The trailer

bytes 1 and 2 - Frame Check Sequence:

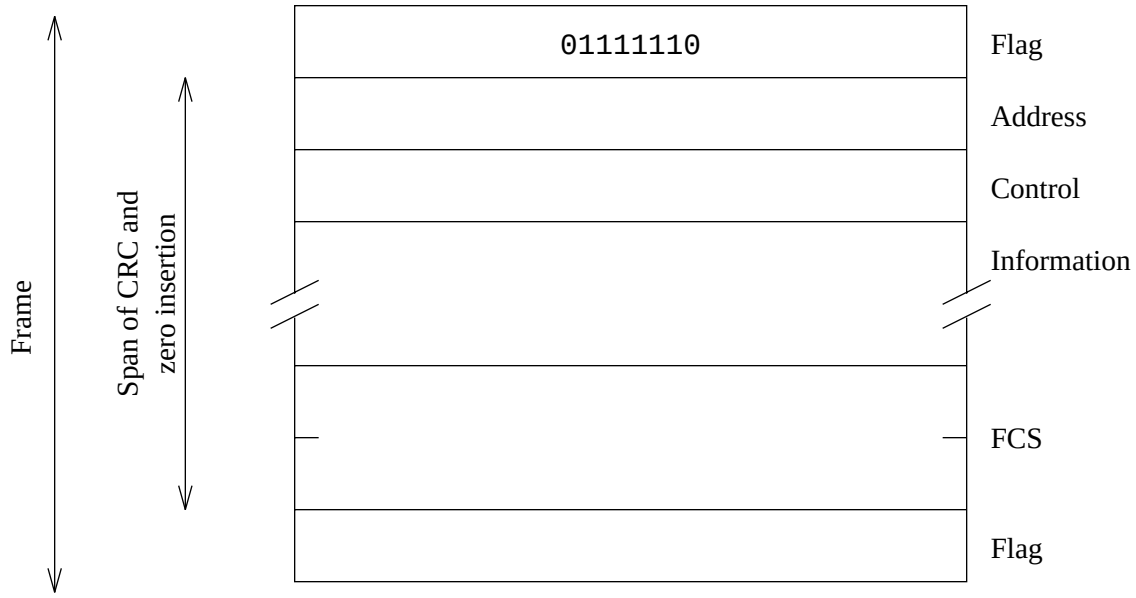


Figure 2.10. The HDLC frame format

The FCS field contains a 16-bit Cyclic Redundancy Check (CRC) error detecting code.

byte 3 - Flag:

This flag indicates the end of the frame. When this flag is detected the receiver examines the preceding two bytes and executes its error detection algorithm.

A flag may be followed by a frame, by another flag, or by an idle condition.

2.6.4. Data link channel states

Active channel state:

A channel is in an ACTIVE state when the primary or a secondary is actively transmitting a frame, a single abort sequence or interframe time fill. In this state the right to continue transmission is reserved.

Abort:

A station can abort a frame by transmitting at least seven contiguous ones. The receiving station will ignore the frame.

Interframe time fill:

In this state continuous flag bytes are sent between frames. (See figure 2.11).

Idle channel state:

A channel is defined to be in an IDLE state when the transmission of 15 or more contiguous one bits is detected. In this state a primary must repoll a secondary before transmitting an I-frame to it.

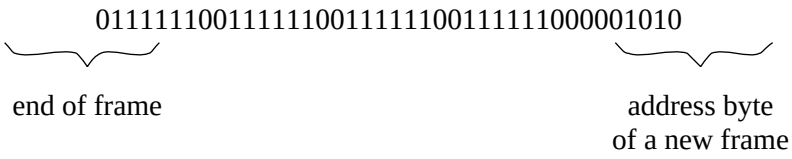


Figure 2.11. Interframe time fill

2.6.5. Types of frames

There are three kinds of frames:

I-frame: Information frame: This type of frame carries user's data.

S-frame: Supervisory frame: This is used for supervisory control functions such as acknowledgements, requesting transmission and requesting a temporary suspension of transmission.

U-frame: Unnumbered frame or Unsequenced frame: This is used to provide additional link control functions.

The contents of the control field for these three types are shown in Figure 2.12.

Before discussing in detail the functional aspects of these frames we will look at the modes of operation.

2.6.6. Modes of operation

For a secondary station two operational modes are defined: Normal response mode and Asynchronous response mode.

Normal response mode

In this mode a secondary station can transmit only in response to a command frame from the primary station. The response transmission may consist of one or more frames, the last of which must be explicitly indicated (using the Poll/Final bit). The secondary then cannot transmit again until it receives another command to do so.

Asynchronous response mode

In this mode the secondary station may initiate transmission of a frame or group of frames without receiving explicit permission from the primary. The frames may contain data or control information. The secondary station is responsible for time-out and retransmission if necessary.

ARM is necessary for the control of loops of stations or multi-drop lines with hub polling. In these configurations a secondary may receive a "go-ahead" message from another secondary and transmit in response to it.

Unfortunately, the above two modes are *unbalanced* modes in which one end of the link is the master and the other end is the slave. To make the protocol more suitable when the two stations are equals (e.g. the nodes of mesh structured computer networks), HDLC has an additional mode: Asynchronous Balanced Mode (ABM).

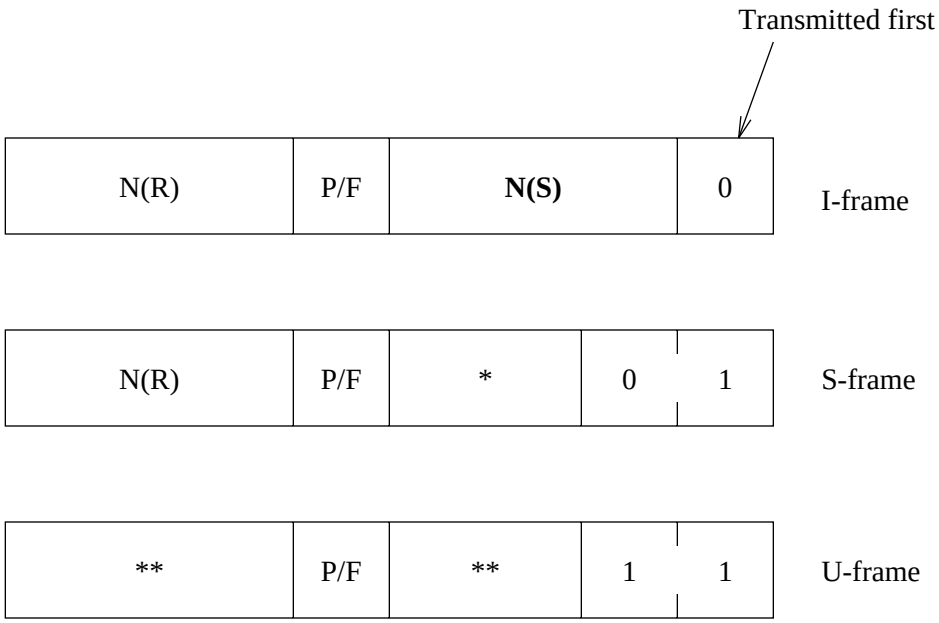


Figure 2.12. Control field of I, S and U frames

Asynchronous balanced mode (or simply Balanced mode)

In this mode the stations have identical protocols. They both can initiate transmission at any time. They can send both commands and responses.

The three modes of operation are illustrated in Figures 2.13, 2.14 and 2.15.

2.6.7. Commands and responses

In this section we will describe the various information, supervisory and unnumbered frames. These types of frame are distinguished by their 8-bit control fields. Within each type, the frames are differentiated by the ways the bits in the C-field are configured. We will look closely at the C-field configurations, their associated functions and frames. When one of the frames is received by a secondary station, it is a command; when it is received by a primary station, it is a response.

I-frames

The format of an I-frame is shown in Figure 2.16.

The I-frames are identified by the first bit sent of the control field. This bit is a 0.

The next three bits provide counts for numbering the frame being sent. It is called the *send sequence number* (N(S)).

The next bit is the Poll/Final bit. It is the send/receive control. A P bit is sent to a secondary station to authorise transmission. An F bit is sent by a secondary station in response to the P bit. Normally, only one P bit is outstanding on the data link.

The next three bits provide counts for numbering the frame expected to be received next. It is a piggybacked acknowledgement, and is called the *receive sequence number* (N(R)).

S-frames

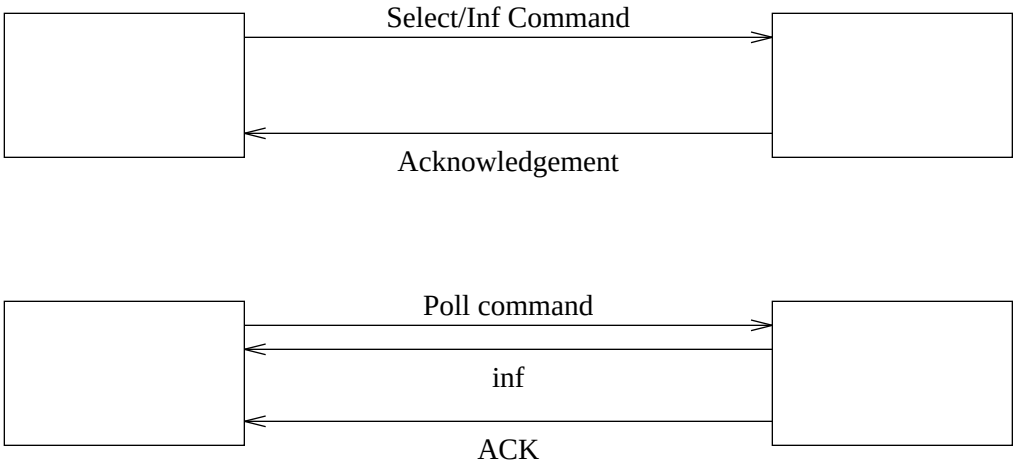


Figure 2.13. Normal response mode

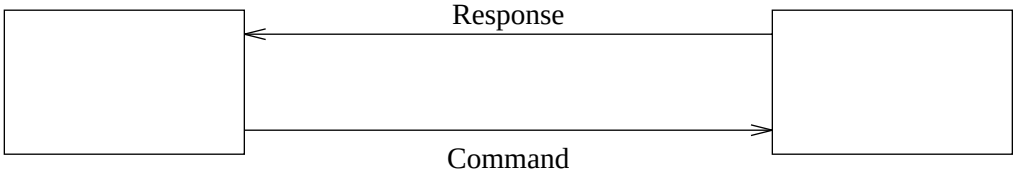


Figure 2.14. Asynchronous response mode

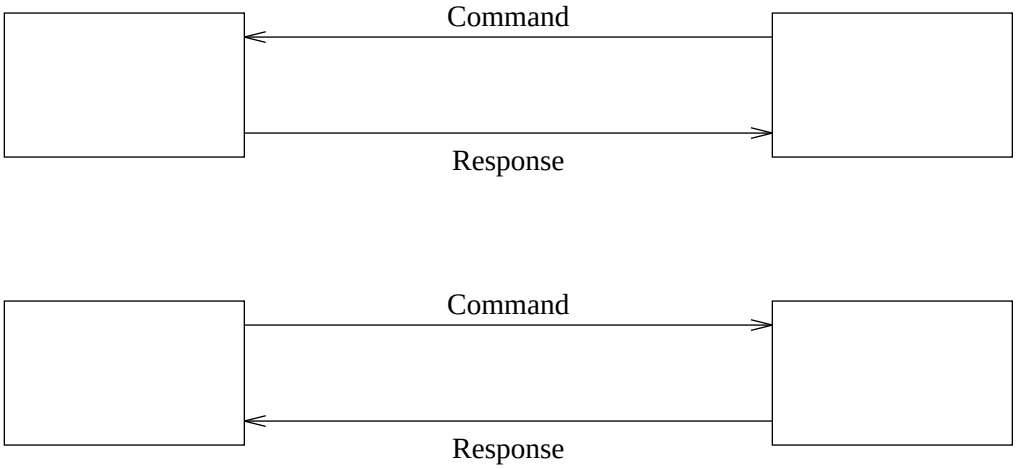


Figure 2.15. Asynchronous balanced mode

The format of an S-frame is shown in Figure 2.17.

The S-frames are identified by the first two bits sent of the control field. These are 10.

The next two bits are used for coding the commands and responses (see Figure 2.17).

The Poll/final bit is as in the I-frame.

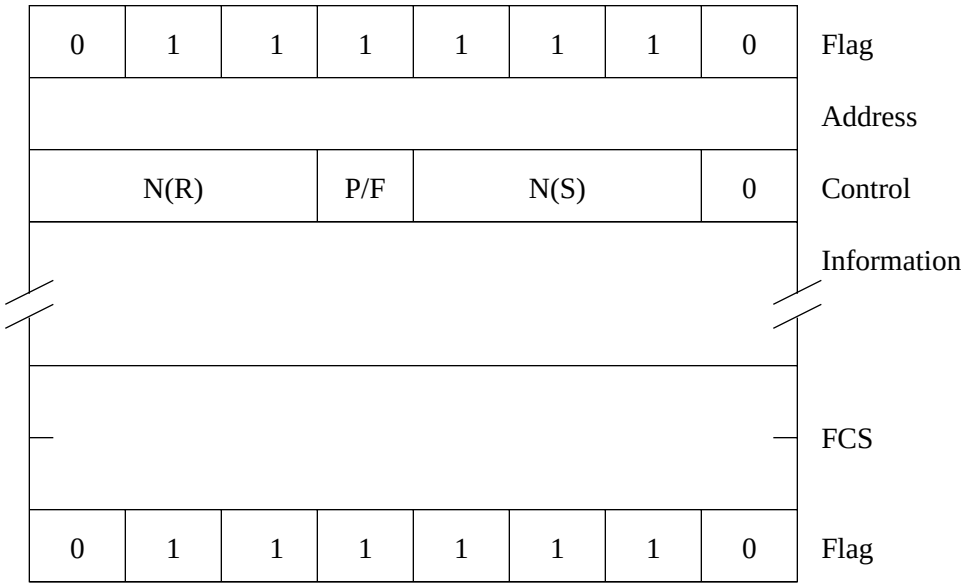
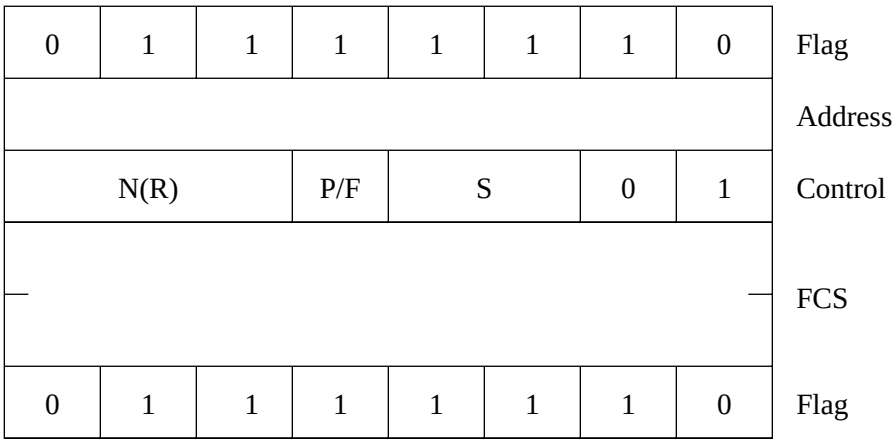


Figure 2.16. The HDLC I-frame format



S = 00: RR Receive Ready (command or response)
S = 01: REJ Reject (command or response)
S = 10: RNR Receive not ready (command or response)
S = 11: SREJ Selective reject (command or response)

Figure 2.17. The HDLC S-frame format

The next three bits form N(R), which indicates that all I-frames numbered up to N(R)-1 have been correctly received and the receiver is expecting the I-frame numbered N(R).

The S-frame does not contain an information field. Consequently it does not have N(S) and it is normally a six byte frame.

The four types of S-frame are described below:

(i) Receive Ready (RR)

An RR frame confirms receipt of frames numbered up to N(R)-1 and indicates that it is ready to receive frame number N(R). An RR frame with the poll bit set to 1 may be used by a primary station to poll a secondary station.

(ii) Reject (REJ)

An REJ frame is used by the primary or secondary station to request retransmission of information frame number N(R) and those that follow it. It implies that I-frames numbered N(R)-1 and below have been correctly received.

Only one reject condition may be established at any given time for a given direction of transmission over the link. The REJ condition is cleared by receipt of an I-frame with an N(S) equal to the N(R) of the REJ command/response frame.

(iii) Receive Not Ready (RNR)

An RNR frame indicates a temporary busy condition. The station which sends the RNR frame acknowledges I frames up to N(R)-1, expects I-frame number N(R), and indicates that it cannot accept any more I-frames. This busy condition is cleared upon the receipt of any other S-frame and of certain types of U-frame.

(iv) Selective Reject (SREJ)

An SREJ frame is used by the primary or secondary to request retransmission of the *single I-frame numbered N(R).* All frames up to N(R)-1 have been received correctly but N(R) has not. Once SREJ has been transmitted, the only I-frames accepted are the frame N(R) and those that follow it.

Figure 2.18 shows an example of retransmissions following a timeout after a packet was received incorrectly.

U-frames

The U-frames are identified by the first two bits sent of the control field. These bits are 11. The P/F bit is used as before. The other five modifier bits are available to encode up to 32 types of U-frame. However, not all 32 combinations are used yet. A U-frame has no sequence numbers. U-frames are used to provide additional link control functions such as

- commands which can initialise a station, change transmission modes, and disconnect a station;
- provide a mechanism for rejecting invalid commands or dealing with some special error conditions.

The control field of the U-frame is shown in Figure 2.19.

The SNRM command allows a station that has just come back on line to announce its presence. The secondary station confirms acceptance of SNRM by transmitting a single UA response frame with the F bit set to one. The secondary station forces all the sequence numbers to zero. The SNRM contains no information field. The SARM and SABM commands function similarly.

The DISC command terminates an operational mode previously set by a command and places

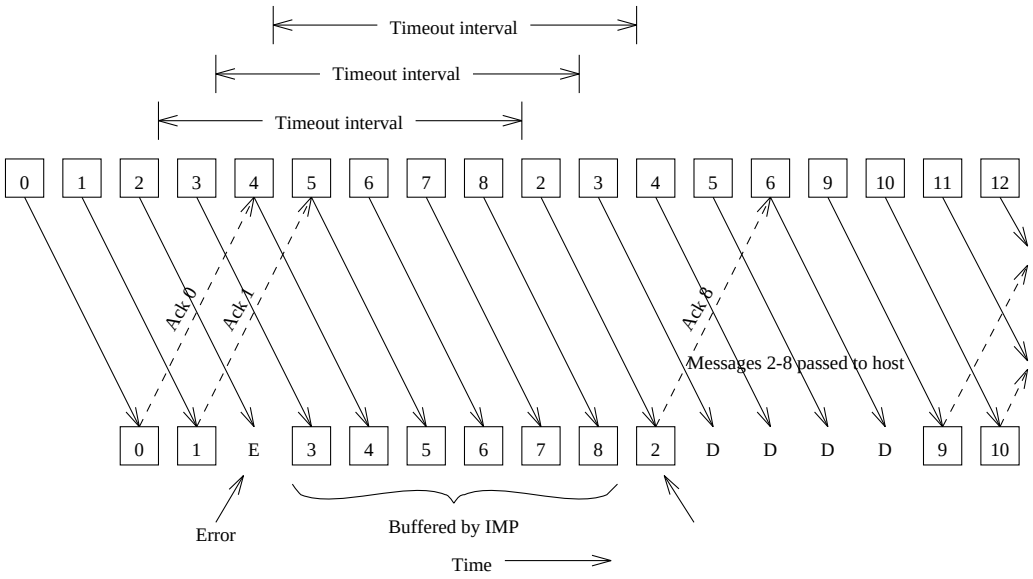


Figure 2.18.

Some typical commands:

SNRM	Set normal response mode
SARM	Set asynchronous response mode
DISC	Disconnect
SABM	Set asynchronous balanced mode

Some typical responses:

UA	Unnumbered acknowledge
CMDR	Command reject

Figure 2.19. U-frame

the receiving secondary station effectively off line. The expected response is a UA frame. DISC contains no information field.

The Unnumbered Acknowledge (UA) frame is used by the secondary station to acknowledge the receipt and acceptance of U-frame commands from the primary. The UA frame contains no information field.

The CMDR response is used by a secondary station when it receives a non-valid frame. A received frame may be non-valid for several reasons:

(i) the command is not implemented at the receiving station;

(ii) the I-field is too long to fit into the receiving station's buffers; or

(iii) The N(R) received is incongruous with the N(S) that was sent to it.

The Command Reject frame has an information field that contains details of why the command was rejected.

2.6.8. Information Exchange

In this section we will describe how information exchange between stations takes place using HDLC protocols. We will also describe methods employed for error recovery in some typical abnormal error conditions (not all possible situations).

Set-up

If a secondary station is off-line, the primary station must first send a command (e.g. SNRM) to the secondary station to set it up in some operational mode (e.g. NRM). The secondary station confirms the acceptance of the command by setting its N(R) and N(S) to zero and returns an unnumbered acknowledge (UA) frame to the primary. Both stations are then ready for information exchange. This procedure is indicated diagrammatically in Figure 2.20.

Polling

Having agreed upon the operational mode, both stations can now exchange information. However, if the mode is normal response mode then

- the primary can send data or command frames to the secondary station at any time, but
- the secondary station can only transmit if it is permitted to do so, i.e. if it receives a frame with the poll bit set to one.

An example is given in Figure 2.21.

Normal Response

When a station receives enough data to build up a frame together with the address of the receiving station it will first compute the Frame Check Sequence (FCS). It will then start transmission by generating a flag, serialise the address bits, the control bits and data bits, followed by the FCS and the final flag.

The receiving station constantly monitors the line and searches for a flag. When a flag is detected the receiver knows that a frame is on its way, and checks the address byte received to see whether the frame is destined for it or not. If it is, the receiver then continues to receive all the bits of the frame until another flag is seen. The receiver then computes the FCS and compares it with that received. If the frame is correctly and properly received, the receiving station will send an acknowledgement (ACK) message to the sender at the earliest opportunity. The ACK message could be a separate message or it could be piggybacked on an I-frame travelling to the original sender.

Error control

Error control is achieved at this level by the following procedures:

(i) FCS: Data, address and control bits in the frame are CRC coded so that transmission errors can be detected effectively.

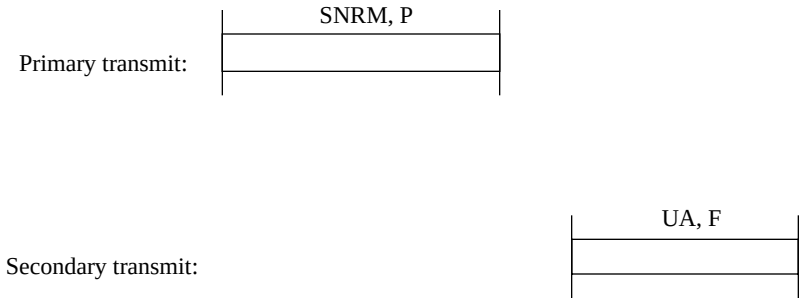


Figure 2.20.

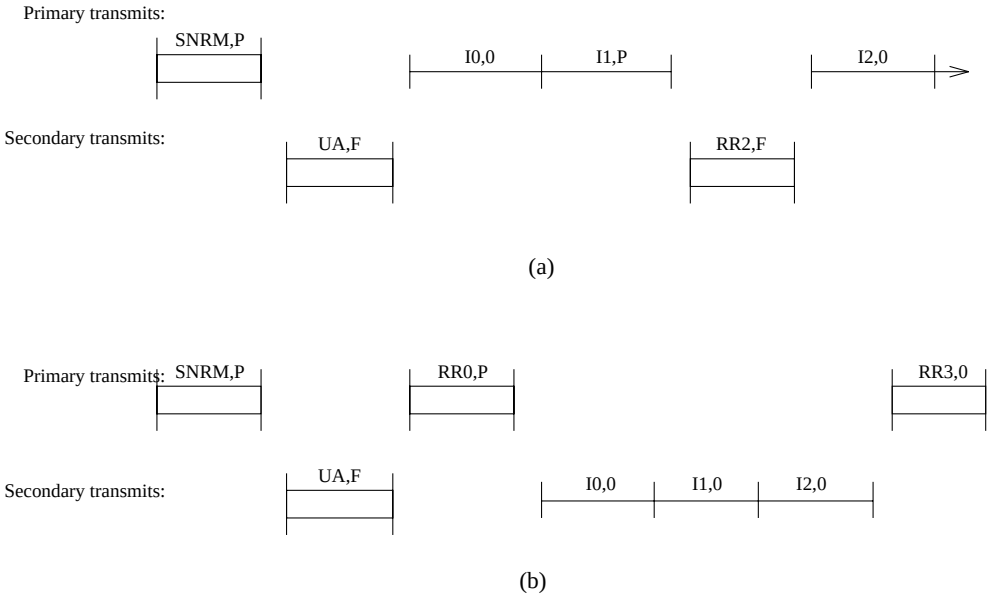


Figure 2.21.

(ii) N(S) and N(R) are also used for error control. If the received frame whose send sequence number N(S) does not match the expected frame number N(R) the receiver knows that an error has occurred, i.e. some frame is missing or the received frame is a duplicate.

(iii) The transmitter can time out and retransmit an unacknowledged frame or the receiver can request the retransmission of a frame or group of frames.

(iv) The Poll and Final bits could also be used for error control purposes.

Flow control

Flow control is achieved in various ways:

(i) HDLC is a pipelining protocol. The transmitter is allowed to transmit a number of frames up to $N_{\max}$ before an ACK is received. This is different from the stop-and-wait procedure in which every frame must be acknowledged before the next frame is allowed to be transmitted. The pipelining protocol thus

- speeds up the flow of information; i.e. it is more efficient; and
- prevents the receiver from being flooded.

(ii) Go-back-N and selective reject requests by the receiver also improve the flow.

(iii) The supervisory frames Receive Ready and Receive Not Ready are used to regulate the flow.

Chapter 3. The Network Layer

3.1. Routing Techniques in Computer Communication Networks

3.1.1. Introduction

A computer network consists of computers and communications resources and users. The resources are there to be shared, and therefore coordination is required among the users for their efficient and fair use. This coordination is obtained via network protocols, which govern the access to resources, the establishment of connections and the exchange of data through the network.

Information data (often in the forms of packets) to be transferred from a source (an entry point to the network) to a destination (an exit point of the network) generally pass through many intermediate nodes in the subnet. How the packets are routed within the subnet is the topic of this section.

The goal of the routing protocol is to provide the best collection of paths between source and destination, given the traffic requirements and the network configuration. *Best path* usually means path of minimum average delay through the network, although many other performance criteria could be considered equally valid.

The routing procedure is part of the layer 3 protocol, which is the layer just above the data link control layer (which ensures correct transmission and reception of packets between any *neighbouring nodes* in a network).

Routing techniques have received a great deal of attention in recent years. They have been variously classified as deterministic or stochastic, fixed or adaptive, centralised or distributed. We will look at the classification later.

Regardless of how the routes are chosen for packets, a routing procedure should have certain desirable properties such as: correctness, simplicity, robustness, stability, fairness and optionality.

3.1.2. The Routing Problem

We assume that a traffic pattern $r(i,j)$ (packets per second) from source i to destination j, $i = 1, \ldots, N; j = 1, \ldots N$, where N is the number of switching nodes, is presented to the network. The objective of the routing procedure is to transport packets on minimum-delay paths from source to destination. Under appropriate assumptions, the average delay T of a packet travelling from source to destination (the average is over time and over all pairs of nodes) is given by

$$T = \frac{1}{\gamma}\sum_{i=1}^{b}\frac{f_i}{C_i - f_i}$$

where

$$N = \text{number of nodes}$$

$$b = \text{number of directed links}$$

$$r_{ij} = \text{average packet rate from source } i \text{ to destination } j \text{ (packets/s)}$$

$$\gamma = \sum_{i=1}^{N}\sum_{j=1}^{N} r_{ij} = \text{total packet arrival rate from external sources}$$

$$f_i = \text{total flow on channel } i \text{ (bits/s)}$$

$$C_i = \text{capacity of channel } i \text{ (bits/s)}$$

The routing problem can be defined as the problem of finding the best routing policy which minimises the average delay T. We may formulate the problem as follows:

Given:

Topology
Channel capacities C_i
Requirement matrix R

Minimise:

$$T = \frac{1}{\gamma}\sum_{i=1}^{b}\frac{f_i}{C_i - f_i}$$

Subject to:

(a) f is a multicommodity flow satisfying the requirement matrix R

(b) $f \leq C$

The static (or fixed) versus adaptive classification is vague since all networks provide some type of adaptivity to accommodate topological changes and to changes in the user traffic

requirement.

If network topology is not subject to changes, and traffic inputs are stationary then the optimal routing solution to the problem stated above is a *static solution* consisting of a set of fixed paths between all node pairs.

In dynamic (or adaptive) routing, ideally every time there is a change in topology and/or traffic input requirement an *optimum static solution* is computed. However, this process involves a lot of overhead and is thus time consuming, especially if the changes occur too fast. Often a "recursive solution" subject to some additional requirements is adopted:

1. At steady state, the adaptive routing must converge to the optimal static routing corresponding to the existing traffic pattern.

2. In time-varying traffic conditions, the policy must adjust to load fluctuations so as to yield minimum delay.

3. After failures, the policy must minimise the time to recover to a stable condition.

We will present a number of solution techniques in a later section. In the next section we will try to classify various routing procedures in such a way that the relative position of a routing procedure we can pin-point its main characteristics.

3.1.3. Routing Algorithm Classification

Instead of labelling routing algorithms with a string of various terms, we classify them by associating each routing algorithm with a point in 3-dimensional space as in Figure 3.1.

The x direction shows the place where routing decisions are made, either at the nodes in distributed (D) fashion or centrally (C).

The y direction describes the kind of strategy to be used in the routing algorithm, either invariant (I, or fixed or static) or adaptive (A). This axis measures the speed with which the algorithm can change or adapt.

The third dimension describes the kind of information to be used in making routing decisions, either local (L), i.e. use of only the information locally available at the nodes or global (G) information.

We will see in various realisable algorithms that where the routing decisions are made is most important; the strategy and the information available will then very much depend on traffic patterns, topological changes and the amount of overhead involved.

We will define *routing table* and *functions of routing procedures* in the next sections before presenting solution techniques.

3.1.4. Routing Table Representation

A fixed routing procedure is represented by a set of routing tables, one for each node, indicating how the packets arriving at that node must be routed on the outgoing links, depending on its final destination.

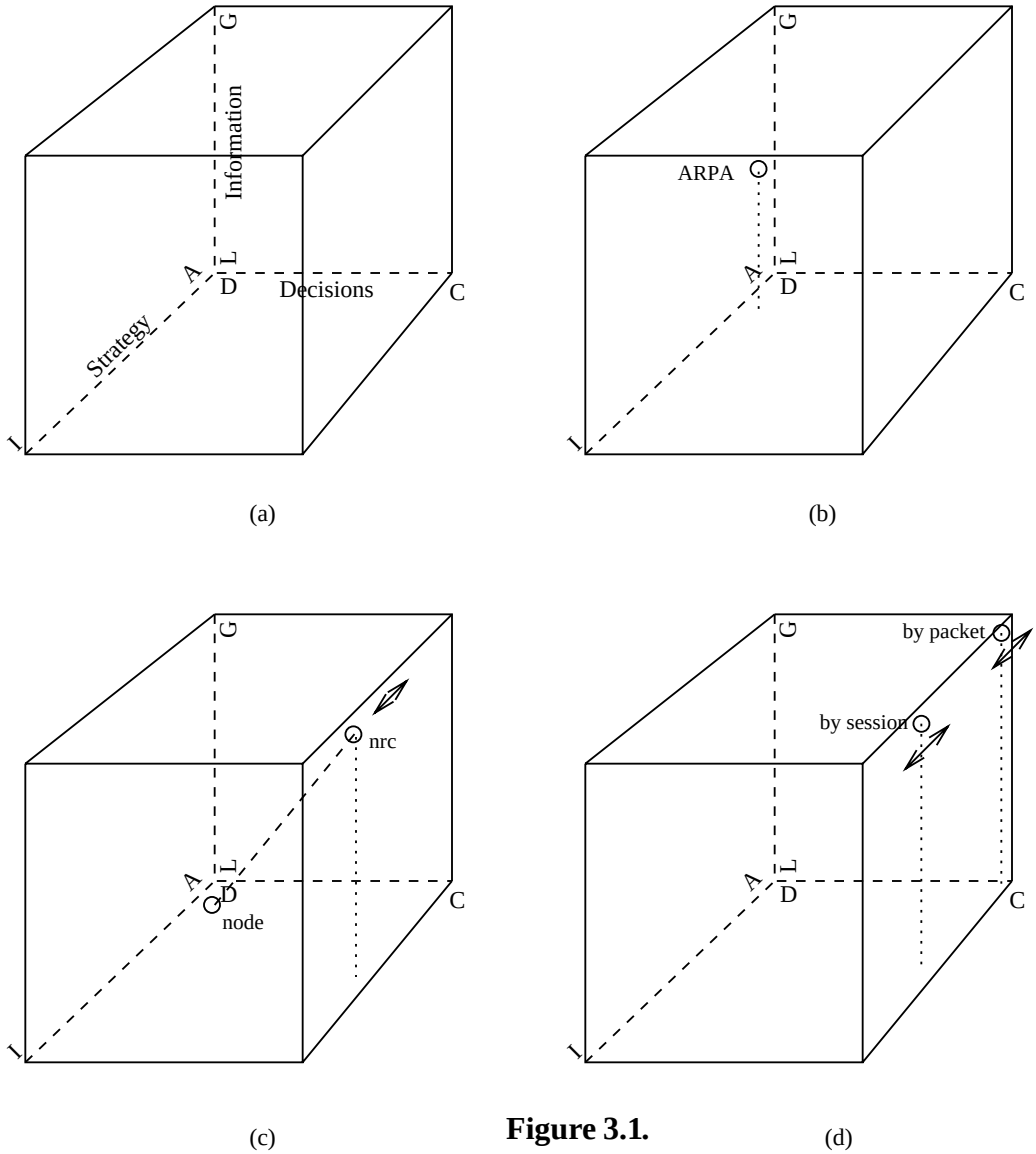


Figure 3.1.

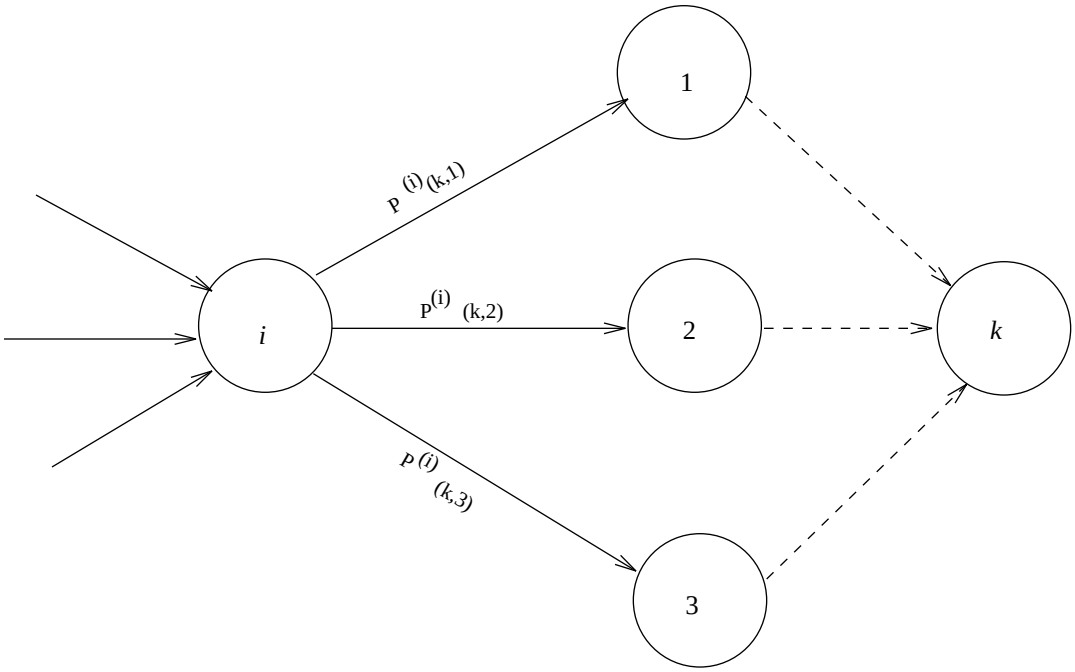
In adaptive routing procedures the routing tables are updated according to the changes in the network.

The most general routing table for node i is an $N \times A_i$ matrix $p^{(i)}(\cdot, \cdot)$, where N = number of nodes in the network and A_i = number of neighbours of node i. $P^{(i)}(k,j)$ is the fraction of traffic directed to k which, upon arrival at node i, is routed through neighbour j (Figure 3.2).

For adaptive routing procedures, the entries $P^{(i)}(k,j)$ in the table will vary with time.

In general a routing procedure tries to minimise the average delay from source to destination. A delay table similar to the routing table is made up and the formation of a simple routing table is as follows (Figure 3.3).

For any given destination k the minimum entry in the delay table provides the appropriate outgoing link over which to route messages destined for k. These are shown circled. The corresponding output lines appear in the routing table of Figure 3.3(c).



k	1	2	...	A_i
	$P^{(i)}(k,1)$	$P^{(i)}(k,2)$	...	$P^{(i)}(k,A_i)$

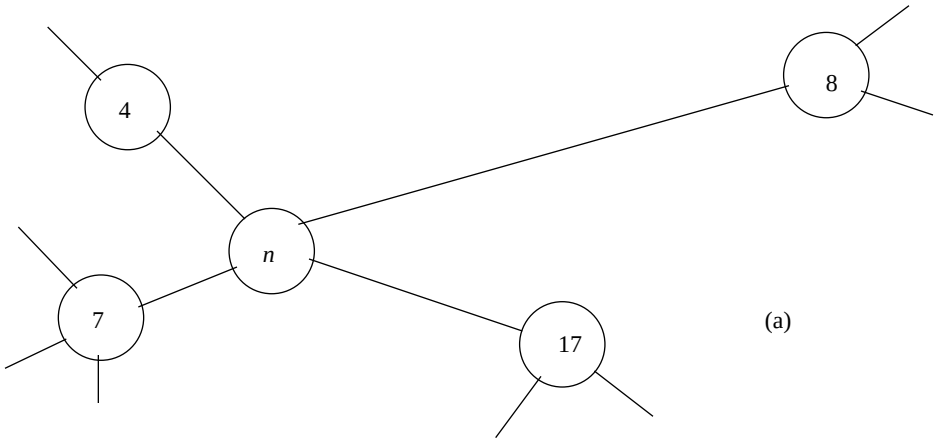
Figure 3.2. Routing Table $P^{(i)}$ at node i

3.1.5. Functions of Routing Procedures

If all the parameters of the network are known and not changing then the routing procedure is quite simple: a routing strategy which optimises the network performance is determined once and for all; optimised fixed paths between any source and destination are stored at each intermediate node and packets are forwarded accordingly.

In real networks changes such as line failure and variation in the traffic distribution do occur and this necessitates some degree of adaptivity. Any adaptive routing procedure must perform a number of basic functions:

- measurement of network parameters,
- reporting of network parameters,
- route computation and



(a)

Destination \ Neighbour node	4	7	8	17
1	8	12	9	9
2	17	13	9	10
3	18	13	24	17
⋮	⋮	⋮	⋮	⋮
k	11	9	15	11
⋮	⋮	⋮	⋮	⋮
N	16	19	14	12

(b)

Destination	Next node	Est. delay
1	4	8
2	8	9
3	7	13
⋮	⋮	⋮
k	7	9
⋮	⋮	⋮
N	17	12

(c)

Figure 3.3. Formation of the routing table

- route implementation.

Measurement of network parameters

The network parameters pertinent to the routing strategy are the network topology, the traffic pattern, and the delay. Typical parameters consist of states of communication lines, estimated traffic, link delays and available resources. In various routing procedures the network information is distinguished into local, global and partial information. Local information measured by a node consists of queue lengths, flow on outgoing links and external traffic to the local node. Global information consists of the full network status collected from all nodes in the network. Partial information is the status information of the network *as seen by the node*; it involves some exchange information between neighbouring nodes.

Reporting of the network parameters

Network parameter measurements are forwarded to the Network Routing Centre (NRC) in a centralised system and to various nodes in a distributed system for processing. In the distributed

case two alternatives are possible:

- The local measurements are combined with the information received from neighbours to update the partial status of the network.
- Global network information is distributed to all nodes.

Route computation

Based on the measured information, routes with minimum cost are assigned to each possible source-destination pair. In *single path* routing policy all traffic for a given source-destination pair follows the same path. In multipath (or bifurcated) routing policy traffic for a given source-destination pair might be distributed over several paths.

Route implementation

At each node a routing table is stored. Packet forwarding is done at each node by inspecting the packet header of each incoming packet, consulting the routing table and placing the packet on the proper output queue. Virtual circuit and datagram are two forwarding mechanisms often employed in a real network.

3.1.6. Solution Techniques

Centralised Solution

Consider a network with N nodes and b links. The problem is to minimise the average delay T of equation (1). We first define the following notation:

$f_i^{(m)} =$ the average flow in link i caused by commodity m, i.e. caused by the source-destination (S-D) pair m.

$f_i = \sum_{m=1}^{M} f_i^{(m)}$, if there are M S-D pairs.

and at each node l we define

$f_{kl}^{ij} =$ the flow from node k into node l caused by S-D pair (i,j).

$f_{lq}^{ij} =$ the flow out of node l to node q caused by S-D pair (i,j).

Consider a typical node l. "Message flow conservation" states that the total average flow into node l due to commodity m must be equal to the total average flow out of node l due to commodity m, i.e. for each commodity m (say S-D pair (i,j))

$$\sum_{k=1}^{N} f_{kl}^{ij} - \sum_{q=1}^{N} f_{lq}^{ij} = \begin{cases} -r_{ij} & \text{if} l = i \\ r_{ij} & \text{if} l = j \\ 0 & \text{otherwise} \end{cases}$$

For M commodities there are $M \times N$ such operations. For a full duplex, fully distributed network, $M = N(N-1)$; there are $N^2(N-1)$ such equations. The object of the routing strategy is to find each f_{kl}^{ij} in the network, subject to the constraints stated in section 1.2.

Various optimal techniques for solving this multicommodity (MC) flow problem are found in the literature; however, their direct application to the routing problem proves to be cumbersome and computationally inefficient.

One iterative technique that has been found very useful is the *Flow Deviation (FD) method.*

Flow Deviation Method

For a given MC flow $\underline{f}$, let us define link length as a function of link flow of the form

$$l_i = \frac{\partial T}{\partial f_i}, i = 1, 2 \ldots b.$$

The FD algorithm is as follows:

Algorithm:

(1) Let $\underline{f}^n = (f_1, f_2, \ldots f_b)$ represent the current (n^{th} iteration) set of link flow, compute the shortest route flow vector $\underline{v}^n$ associated with link lengths $(l_1, l_2, \ldots l_N)$ satisfying the MC constraints.

(2) Set

$$\underline{f}^{n+1} = (1-\lambda)\underline{v}^n + \lambda f^n; 0 < \lambda < 1$$

and find an optimum λ such that $T(f^{(n+1)})$ is minimised.

(3) If $|T(f^{n+1}) - T(f^n)| < \varepsilon$, stop. Otherwise, let $n = n + 1$ and go to step (1).

The FD technique provides the *total* flow f_i in any link i. To determine the individual commodity flows in each link, as required for routing assignment, additional bookkeeping must be carried out.

The individual commodity flow in link i by commodity m, $f_i^{(m)}$ is a by-product of the FD algorithm. From these $f_i^{(m)}$ the routing tables are derived as follows:

$$P^{(i)}(m,j) = \frac{f_{i,j}^{(m)}}{\sum_{l=1}^{A_i} f_{i,l}^{(m)}}$$

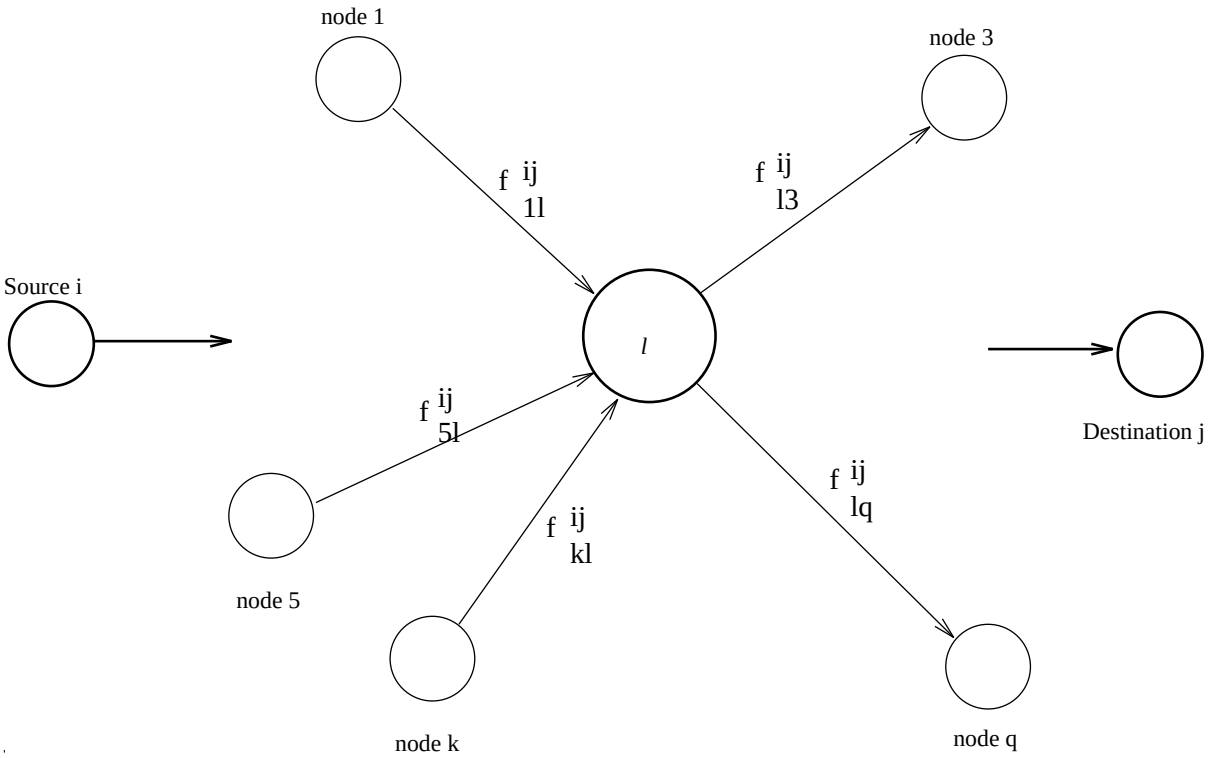


Figure 3.4.

where (i,j) indicates the directed link from i to j and A_i is the number of neighbours of node i.

The FD algorithm has been applied to various examples of ARPA-type network. However, other algorithms have been suggested as well.

Distributed Solution

In this section we show that the optimal solution can also be obtained via a distributed process, where each node in the network participates in the routing computation. We begin with a simple shortest path routing problem before considering the minimum-delay routing problem.

Shortest path routing problem

For an arbitrary, connected topology, $l(i,j)$ is defined to be the length of link (i,j). We wish to find the shortest paths from each node to all destinations.

We assume that at each node $i(i = 1, \ldots N)$

(a) there is an $N \times A_i$ matrix $DT^{(i)}$, called a *distance table*, whose entry $DT^{(i)}(k,n)$ is the estimated minimal distance from node i to destination k if n is chosen to be the next node in the route to k;

(b) an N-dimensional minimum distance vector $MDV^{(i)}$, which represents the minimum estimated distance from i to k, is derived as follows:

$$MDV^{(i)}(k) = \min_{n \in A_i} DT^{(i)}(k,n)$$

In the distributed shortest path procedure each node periodically updates its distance table by using the knowledge of the length of its outgoing links and the information received from its neighbours. During every interval δ, each node exchanges the vector MDV with its neighbours. When node i receives the vector $MDV^{(n)}$ from its neighbour n, it updates $DT^{(i)}$ as follows:

$$\begin{cases} DT^{(i)}(k,n) = l(i,n) + MDV^{(n)}(k) & ; k \neq i \\ DT^{(i)}(k,n) = 0 & ; k = i \end{cases}$$

with

$DT^{(i)}(k,n) = \infty$ initially; for all k,n

This procedure, repeated for a finite number of steps, leads to a final set of tables $DT^{(i)}(k,n)$, where all the entries are $<$ inf. The vector $MDV^{(i)}(k)$ calculated from the final table DT is the shortest distance from i to k and the neighbour $n*$ such that

$$DT^{(i)}(k,n*) \leq DT^{(i)}(k,n) \textit{for all } n \in A_i$$

is the next node on the shortest path from i to k.

If $l(i,j) = 1$ for all the links in the network, the algorithm becomes the well-known minimum-hop routing algorithm. .uh The minimum delay routing algorithm

The minimum delay routing distributed algorithm is due to Gallager. The link length is defined to be

$$l(i,j) = \frac{\partial T}{\partial f_{ij}} = \frac{1}{(C_{ij} - f_{ij})^2}$$

$l(i,j)$ approximates the incremental delay due to a unit increment of flow on link (i,j). At each node $i(i = 1,2,\ldots,N)$ we assume

(a) There is a matrix $IDT^{(i)}(k,n)$ similar to $DT^{(i)}(k,n)$, called the incremental delay table.

(b) The routing table $P^{(i)}(k,n)$.

(c) The incremental delay vector $IDV^{(i)}(k)$, which represents the incremental delay produced by injecting a unitary increment of k-traffic into node i, is computed as follows:

$$IDV^{(i)}(k) = \sum_{n \in A_i} P^{(i)}(k,n) IDT^{(i)}(k,n)$$

Each node periodically exchanges *IDV*s with its neighbours and the $IDT^{(i)}$ tables are updated as follows:

$$IDT^{(i)}(k,n) = l(i,n) + IDV^{(n)}(k)$$

The routing tables $P^{(i)}$ are also updated. Letting $n*$ be the minimiser of $IDT^{(i)}(k,n)$ for $n \in A_i$,

$$P^{(i)}(k,n*) = P^{(i)}(k,n*) + \delta'$$

$$P^{(i)}(k,n) = P^{(i)}(k,n) - \delta" for n \neq n*$$

where δ' and $\delta"$ are small positive quantities properly selected so as to satisfy the constraint

$$\sum_{n \in A_i} P^{(i)}(k,n) = 1$$

and

$$P^{(i)}(k,n) > 0$$

In order to prevent looping during the update process, the updating must start from the destination node and propagate back to the origins. This procedure is shown to converge to the optimal solution found by the FD method.

3.2. The Network Layer in X.25

It is believed that packet switching is an appropriate technology for public data networks (PDNs). It is also recognised that the commercial viability of these networks depends largely on the development and adoption of standard access protocols. These standards would facilitate the connection of varying types of data terminal equipment (DTEs) to the various public networks, as well as facilitate international internetworking.

One of the most important standards for the computer industry is CCITT Recommendation X.25. X.25 defines the relationship between a user machine (DTE) and the public data network's equipment, called Data Circuit terminating Equipment (DCE) as shown in Figure 3.6.

3.2.1. General Description of the X.25 Protocol

The X.25 user/network interface consists of three distinct layers of control procedures, consistent with the bottom three layers of the OSI model:

Layer 1 The physical layer specifies the plug and wires for establishing a physical

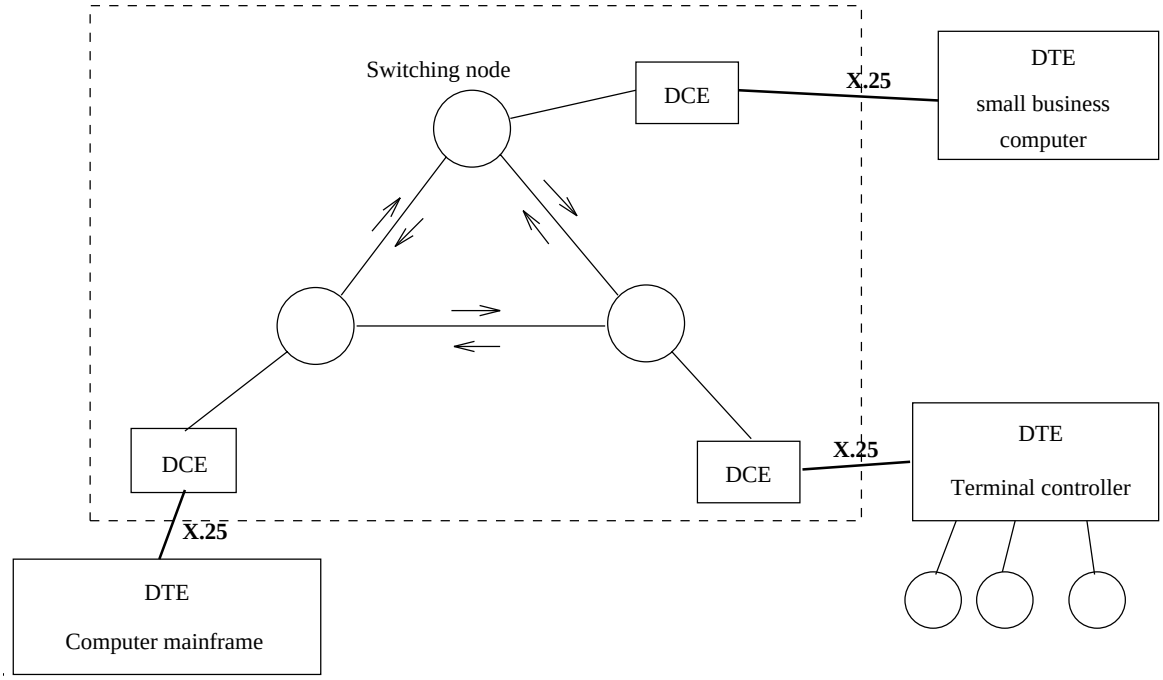


Figure 3.5.

circuit for sending data bits between the user machine (DTE) and the network (DCE). Its functions are to pass data, synchronisation and control signals between the DTE and the DCE and to handle failure detection and isolation procedures. The recommended form of this interface to digital circuits is described in CCITT Recommendation X.21.

Layer 2 — The frame layer is essentially the HDLC layer of the control protocol described previously. In X.25 the control procedures for point-to-point balanced systems are referred to as balanced link access procedures (LAPB). This layer defines control procedures and the frame envelope, which is used to carry a frame of data over the physical link, and ensure that the frame is not lost or garbled.

Layer 3 — The packet layer is the network layer or logical link control layer. This layer specifies the manner in which control information and user data are structured into packets. It describes the formats of packets that are used for setting up and clearing a virtual call, sending data over virtual circuits, controlling message flow, sequencing, interrupts and recovering from the various problems that might occur. It also allows a single physical circuit to support communications to numerous other DTEs concurrently.

Virtual circuit and Datagram service

The X.25 recommendation provides access to the following services that might be provided on public data networks:

- switched virtual circuits (SVCs) or virtual calls

- permanent virtual circuits (PVCs) and
- datagrams.

A virtual circuit (VC) is a bidirectional, transparent, flow-controlled path between a pair of logical or physical parts. A switched virtual circuit is a temporary association between two DTEs. Like a telephone call there are three phases to a virtual call:

(i) call set-up,

(ii) data exchange and

(iii) call clearing.

A permanent virtual circuit is a permanent association existing between two DTEs, which is analogous to a point-to-point private line. It requires no call set-up or call clearing action by the DTE.

A datagram is a self-contained entity of data containing sufficient information to be routed to the destination DTE without the need for a call to be set up (Figure 3.6).

The PAD Interface

Many terminals transmit characters rather than data packets. An interface machine is thus needed to assemble the terminal's data into packets and disassemble them for X.25 operation. Most common carriers provide such an interface machine. A standard for this interface has been proposed as an extension to the CCITT Recommendation X.25 and is called the PAD (Packet Assembly/Disassembly) interface. CCITT Recommendations X.3, X.28 and X.29 define the CCITT PAD interface. X.3 defines the PAD parameters; X.28 defines the terminal-PAD interface; and X.29 defines the PAD-host computer (DTE) interface.

User machines access to the network might assume one of the set-ups in Figure 3.7.

3.2.2. X.25 End-to-end Virtual Circuit Service Characteristics

Establishment and clearing of a virtual circuit

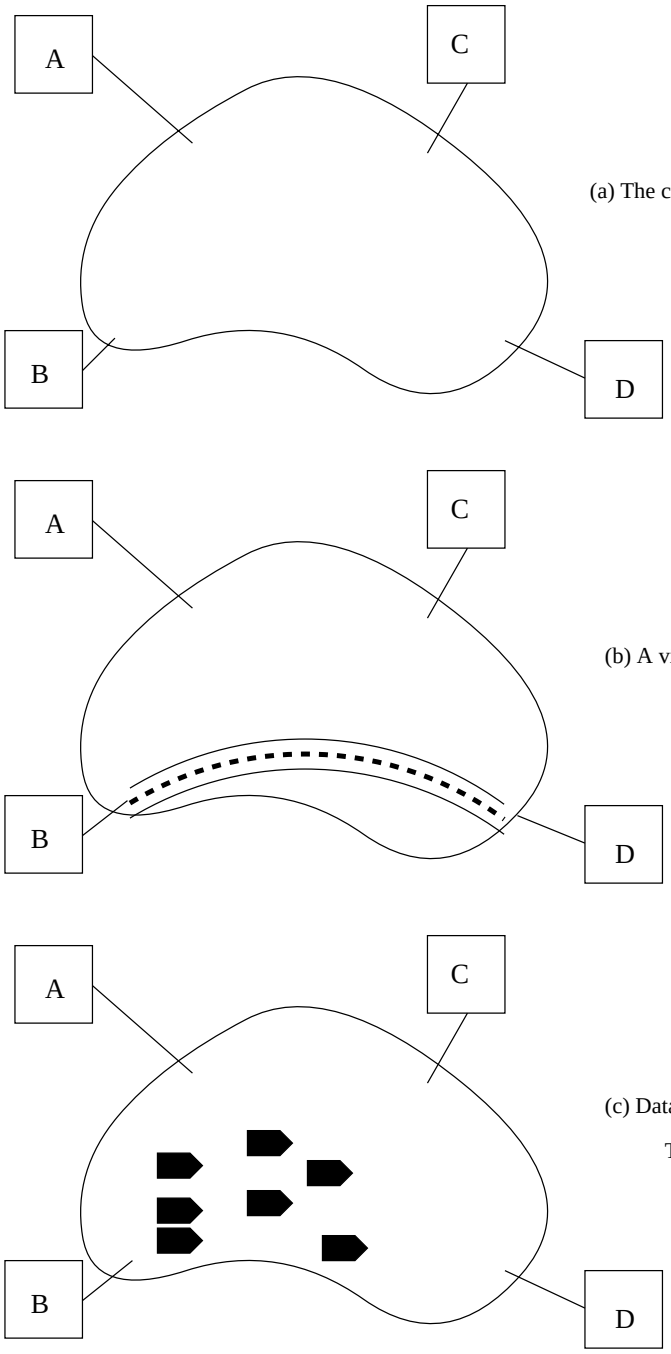
A switched virtual circuit is established when the call request issued by the calling DTE is accepted by the called DTE. The call request identifies the called and calling addresses and facilities requested for the call, and may include user data.

If the call is refused by the called DTE, this DTE can signal the reason for call clearing to the calling DTE in a diagnostic code. If the call attempt fails for some reason, a call progress signal is transmitted across the network indicating one of the causes specified in X.25.

Once the call has entered the data transfer phase, either DTE can clear the call using the diagnostic code to signal to the remote DTE the reason for clearing. If the call is cleared, data may be discarded by the network since the clear is not sequenced with respect to user data.

Data transfer

In the data transfer phase, user data which are conveyed in DATA and INTERRUPT packets are



(a) The communication subnet

(b) A virtual circuit conception

(c) Datagram conception.

The pointy part is the address.

Figure 3.6.

passed transparently through the network. Virtual circuit flow control is applied to ensure that the transmitting DTE does not generate data at a rate that is faster than that which the receiving DTE can accept.

A considerable debate has taken place on whether the DTE or the network should determine the maximum number of data packets which may be in the network on a virtual circuit. It has been agreed that DTE-to-DTE acknowledgment of delivery be available as a standard characteristic

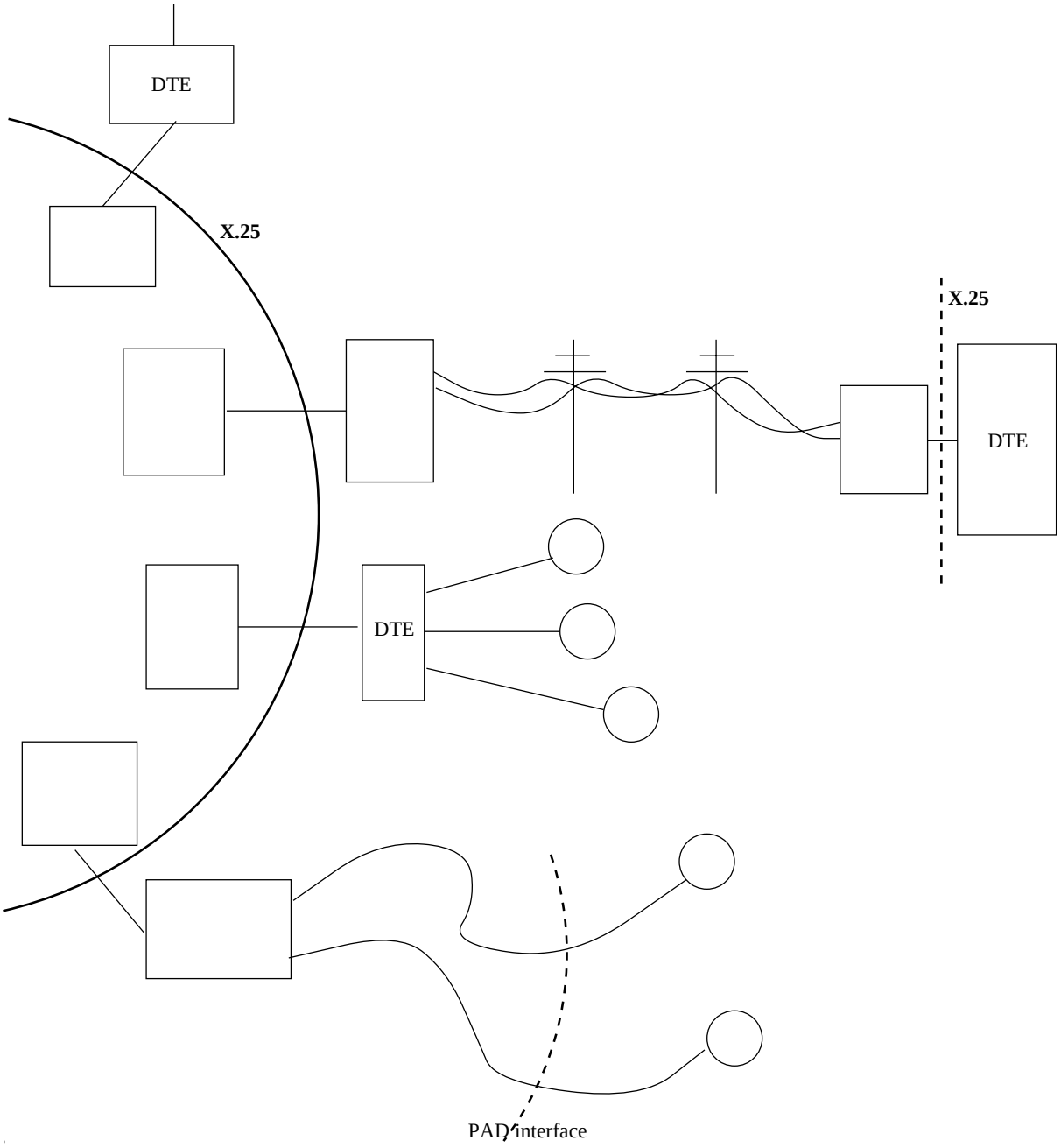


Figure 3.7.

of X.25 virtual circuits. If a DTE wishes to receive end-to-end acknowledgment for data it is transmitting across the X.25 interface, it uses an indicator called the delivery confirmation, or D bit, contained in the header of DATA packets. Later sections will discuss this point further.

There are thus independent mechanisms for transferring user control information between a pair of DTEs outside the normal flow of data or a virtual circuit. The first mechanism transfers user control information within the normal flow control and sequencing procedures in a virtual circuit except that the *Q* bit in the DATA packet header is set. The second mechanism bypasses the the normal DATA packet transmission sequence. This mechanism uses an INTERRUPT packet which may contain one byte of user data. This packet is transmitted as quickly as possible to its

destination, jumping the queues of normal DATA packets.

Packet Format

X.25 describes the formats of packets that will be passed between a user machine and the DCE in order to set up and use virtual circuits. The packets have the general format shown in Figure 3.8.

The first four bits of a packet are a general format identifier. A virtual circuit is identified by the 12-bit *Group* and *Channel* number. When a user machine initiates a virtual call it selects a free logical channel from those available to it. This number is passed to the local DCE which then attempts to set up a virtual call using that logical channel.

There are two types of packets: DATA and CONTROL packets. The *Control* bit is set to 1 in all control packets and to 0 in all data packets.

Figure 3.9 illustrates a CALL REQUEST format, a DATA packet format and various CONTROL packet formats.

3.2.3. X.25 Packet Level Characteristics

This section discusses the X.25 packet level procedures used by DTEs in establishing, maintaining and clearing virtual circuits.

Establishing and clearing a virtual circuit

When a DTE attached to an X.25 network wants to initiate a virtual call, it selects a free logical channel and sends a CALL REQUEST packet to its local DCE. The packet contains the addresses of the destination and originating devices. The addresses may be followed by a facility field of variable length. This field is present only when the DTE wishes to request an optional user facility which must be communicated to the destination DTE. For efficiency, the CALL REQUEST packet may carry up to 16 bytes of user data. The X.25 protocol is unconcerned with the contents of the data field.

The calling DTE will receive a CALL CONNECTED packet (same as CALL ACCEPTED) as a response indicating that the called DTE has accepted the call. The communications between DTEs is illustrated in Figure 3.10.

If the attempt to set up a call is unsuccessful, the DCE responds to the calling DTE by sending a CLEAR INDICATION packet which gives the reason why the call request was not successful.

A DTE may decide to disconnect a virtual call at any time. To do this it sends a CLEAR REQUEST packet to its DCE. The DCE responds when it is ready to clear the channel with a CLEAR CONFIRMATION packet. Figure 3.11 shows the formats of the packets used for clearing.

Data transfer

Once a virtual circuit has been set up, DATA packets can be transferred across the logical channel.

The data field of a DATA packet may be any length up to a maximum of 128 bytes. When a user's data is longer than the maximum packet size, the user divides it up into several packets, which the network delivers in sequence. The third byte of the header contains the *more data* bit (bit *M*) which, if set, indicates that more of the same data record follows in a subsequent packet. The *M* bit can only be set on a maximum length packet.

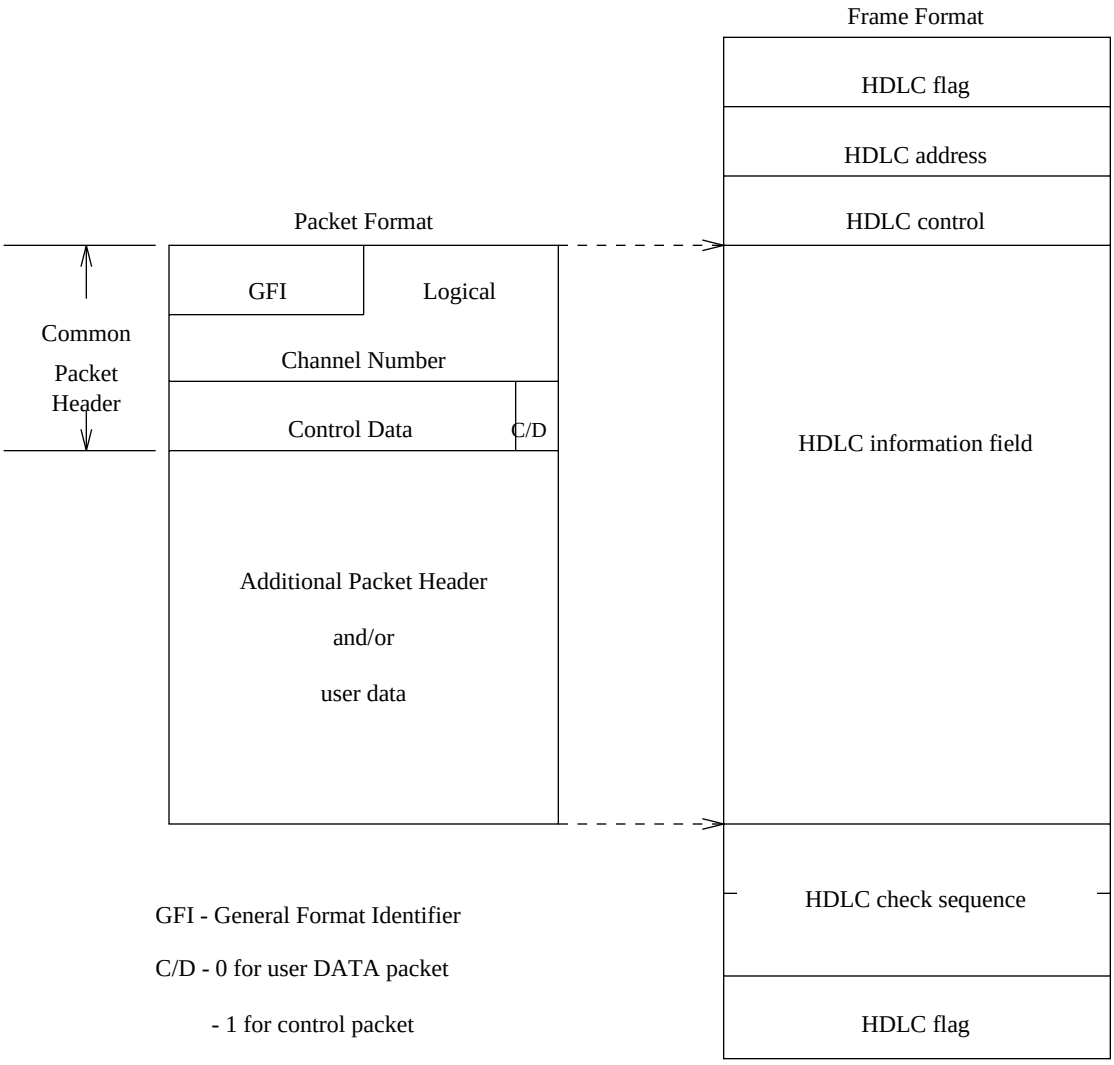
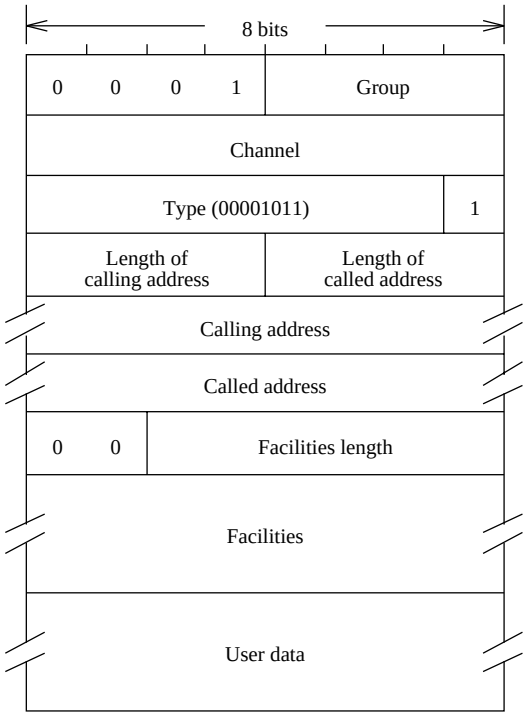


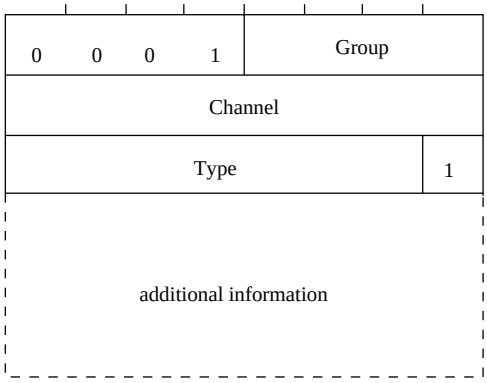
Figure 3.8.

The data packets contain sequential message numbers for flow control. $P(S)$ is the packet send sequence number of the packet, usually modulo 8. The maximum number of sequentially numbered packets that a sender is allowed to send is called the window size W. Each data packet also carries a packet receive sequence number, $P(R)$, which is composed by the receiver when it is ready to receive another packet; this number is the number of the next packet which the receiver expects to receive. The sender can send messages numbered up to but not including $P(R) + W$. This window mechanism for regulating the flow of data is the same as the mechanism used by the HDLC protocol in the data link layer.

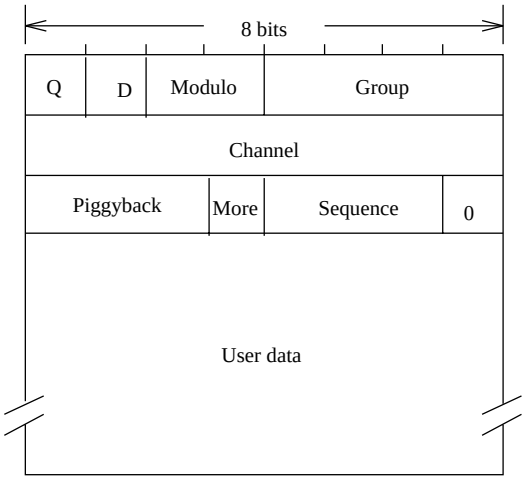
If the DTEs are exchanging data, the flow control signals containing the recieve sequence number can be piggybacked on the returning DATA packets. If not, they must be sent by a separate control message. A RECEIVE READY packet is used to indicate willingness to receive W data packets starting with $P(R)$. On the other hand, a RECEIVE NOT READY packet is returned if the DTE is not ready to receive further DATA packets.



(a) Call request format



(b) Control packet format



(c) Data packet format

Type	Third byte
Data	PPPMSSS0
Call request	00001011
Call accepted	00001111
Clear request	00010011
Clear confirmation	00010111
Interrupt	00100011
Interrupt confirmation	00100111
Receive ready	PPP00001
Receive not ready	PPP00101
Reject	PPP01001
Reset request	00011011
Reset confirmation	00011111
Restart request	11111011
Restart confirmation	11111111
Diagnostic	11110001

(d) Type field

(P = piggyback, S = sequence, M = more)

Figure 3.9. X.25 packet formats

The meaning of the piggyback field is determined by the setting of the D bit. If $D = 0$, the $P(R)$ numbered packet has been received by the local DCE, not by the remote DTE. If $D = 1$, the corresponding $P(R)$ is used to convey an end-to-end delivery confirmation, i.e. that packet has been successfully received by the remote DTE.

An INTERRUPT packet is employed for user control information rather than data. It may be transmitted across the DTE/DCE interface even when DATA packets are being flow controlled. The INTERRUPT packet thus does not contain any sequence number and is confirmed by an INTERRUPT CONFIRMATION packet. Only one unconfirmed INTERRUPT may be

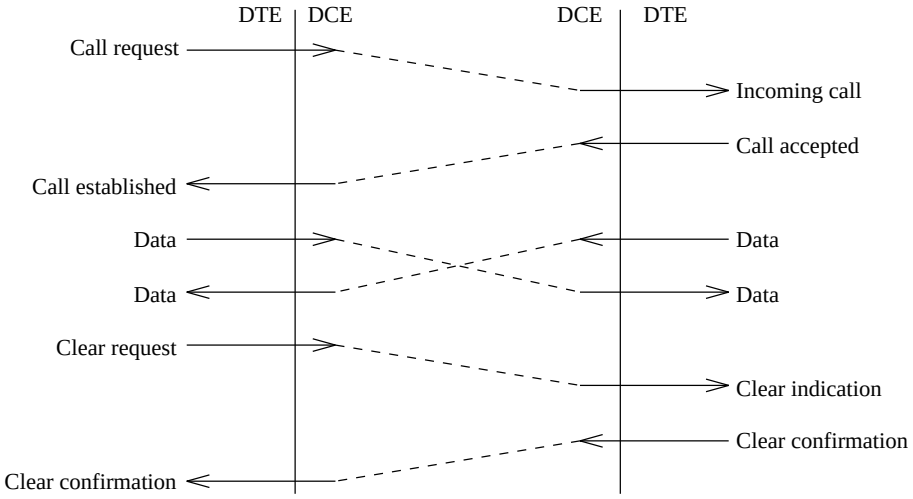
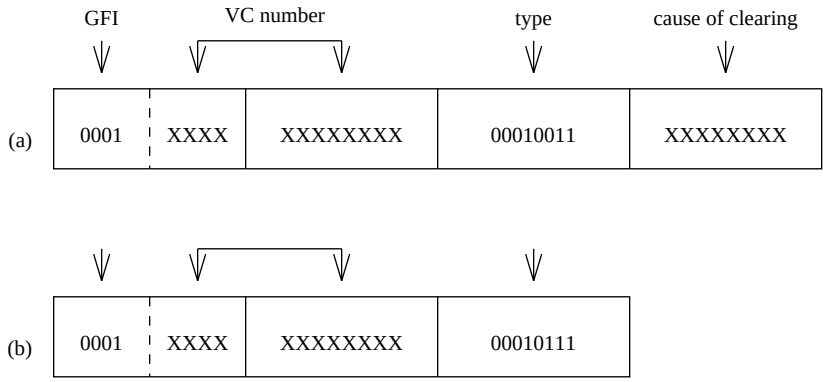


Figure 3.10. Communication between DTEs in X.25



(a) CLEAR REQUEST and CLEAR INDICATION packets

(b) CLEAR CONFIRMATION packets

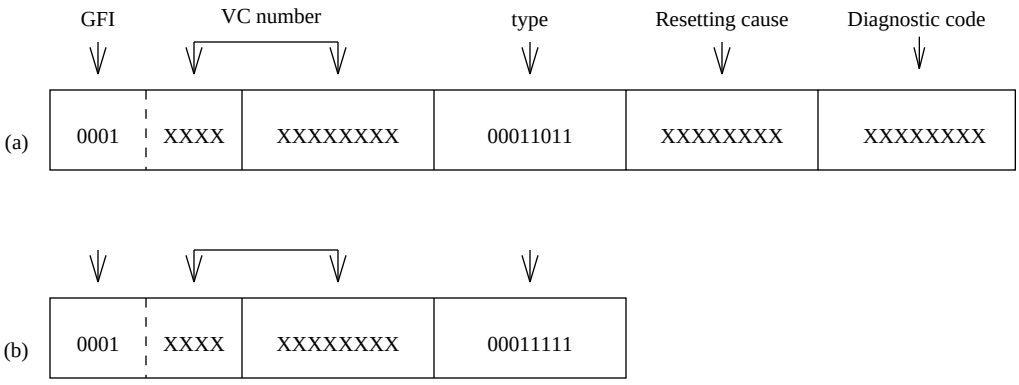
Figure 3.11.

outstanding at any given time. The protocol used for INTERRUPT packets is thus a stop-and-wait protocol, although DATA packets may still be transmitted while waiting for the INTERRUPT CONFIRMATION packet.

3.2.4. Error recovery

Reset

The reset procedure is used to reinitialise the flow control procedure on a given logical channel when certain types of problem occur on the virtual circuit. Any DATA or INTERRUPT packets in transit at the time of the reset are discarded. A reset can be initiated either by a user DTE or by a network DCE. RESET REQUEST and CONFIRMATION packets, illustrated in Figure 3.12, are used in the reset procedure.



(a) RESET REQUEST and RESET INDICATION packets

(b) RESET CONFIRMATION packets

Figure 3.12.

Restart

A restart is equivalent to clearing all the virtual calls that a DTE has connected, and resetting the permanent virtual circuits. The DTE may then attempt to reconnect its calls. The restarting procedure will bring the user/network interface to the state it was in when service was initiated.

Error Handling

The following principles were established in X.25 to handle packet level errors:

(i) procedural errors during establishment and clearing are reported to the DTE by clearing the call;

(ii) procedural errors during the data transfer phase are reported to the DCE by resetting the virtual circuit;

(iii) a diagnostic field is included in the reset packet to provide additional information to the DTE.

(iv) timers are essential in resolving some deadlock conditions;

(v) error tables define the action of the DCE on receiving various packet types in every state of the interface;

(vi) A DIAGNOSTIC control packet is provided to allow the network to inform the user of problems.

3.2.5. A Common X.25 DTE

From a DTE implementation point of view, a common X.25 protocol can be defined, which consists of the following features:

1. an ISO-compatible frame level procedure (i.e. LAPB);
2. use of logical channel number one as the starting point for logical channel assignments;
3. modulo 8 packet level numbering;
4. dynamic *P*(*R*) significance by use of the delivery confirmation bit;
5. a standard procedure for selecting packet and window sizes, with defaults of 128 and 2 respectively;
6. two mechanisms for user control of data transfer: qualified DATA and INTERRUPT packets.

3.2.6. Relationship of X.25 to ISO and CCITT models

The set of commonly agreed standards is collectively known as the Open Systems Architecture (OSA).

It has been generally agreed within ISO and CCITT that the basic structuring technique in OSA is layering. Both ISO and CCITT agree on a seven layer architecture as illustrated in Figure 3.13. A major difference is in the interpretation of the services provided by the network and transport layers and their relationship to X.25 virtual circuits.

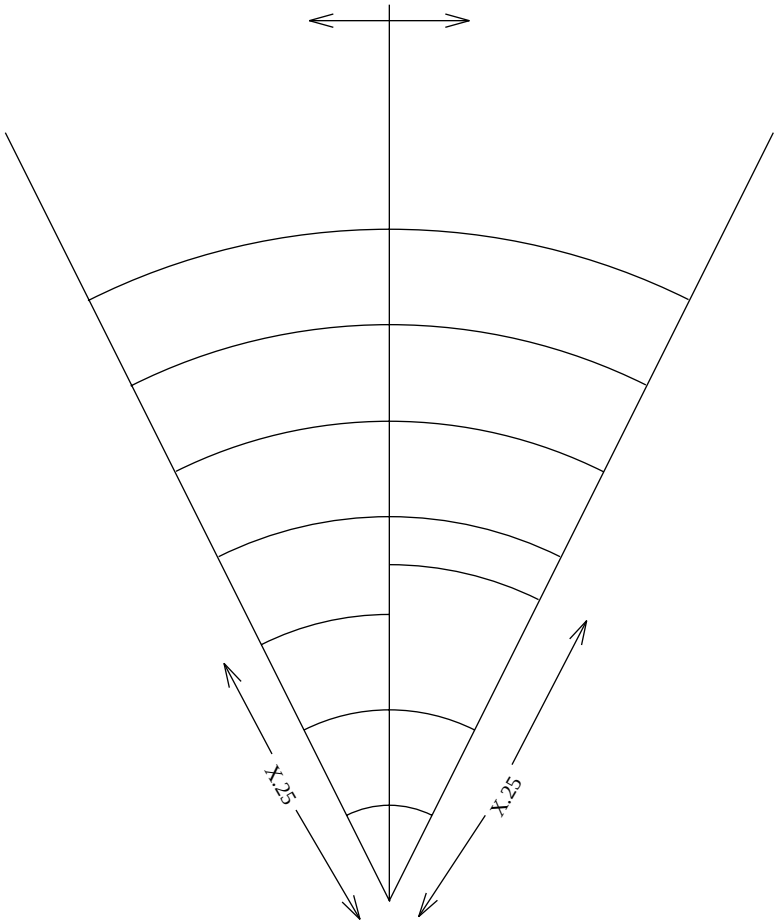


Figure 3.13.

Chapter 4. Packet Protocols for Broadcast Satellites

4.1. Communication Satellites

A communication satellite can be thought of as a big repeater in the sky. It contains one or more *transponders*, each of which covers some portion of the frequency spectrum. The incoming signals (*uplink transmissions*) from ground stations are amplified and then rebroadcast on the *downlink* to these same stations (at another frequency to avoid interference with the upward transmission).

The first satellites had a single spatial beam that covered all ground stations. These stations share the common satellite channel by adopting conventional time division multiplexing (TDM). Synchronization of ground-based transmitters was done by terrestrial cables. Each station received the entire downward transmission, selecting out only those messages destined for it.

Modern satellites are much more sophisticated. Each modern satellite is equipped with multiple antennas and multiple transponders. Each downward beam can be focussed on a small geographical area as small as a few hundred kilometres across, so multiple upward and downward transmissions can take place simultaneously.

To prevent chaos in the sky the frequency bands from 3.7 to 4.2 GHz and 5.925 to 6.425 GHz have been internationally designated for downward and upward transmissions respectively. Other frequency bands suitable are 12/14 GHz and 20/30 GHz, but the equipment needed to use them is more expensive.

Example: SPADE system used on Comsat's Intelsat satellites

Each SPADE transponder is divided into 794 PCM channels, each operating at 64 kbps, plus a single 128 kbps common signalling channel. The channels are multiplexed using Frequency Division Multiplexing (FDM).

The common signalling channel is divided into 50 msec time frames, with each frame containing 50 slots of 1 msec (128 bits). Each slot is permanently allocated to one of the (not more than) 50 ground stations.

When a ground station has data to send, it picks a currently unused channel at random and writes the number of the channel in its reserved 128-bit slot. By the time this request arrives back at the earth and if the selected channel is still unused the requesting station is then allocated the channel. When the station finishes its transmission it sends a deallocation message on the common signalling channel. If two or more stations independently request the same channel, the first request is honoured and the losing stations must try again.

4.2. Satellite Packet Broadcasting

There are two ways of using communication satellites in a data network. They can be used as *point-to-point channels* or as *broadcast channels*. For point-to-point communication the satellites are used in exactly the same way terrestrial cables are used. The satellite is just a big cable in the sky connecting one centre to another. The main advantage of a satellite channel is that it is cheaper for long distances and a high bandwidth can be used, although the use of fibre optic cables has overcome both of these advantages. The main disadvantages are the 540 msec round-trip propagation delay and the need for expensive antennas.

It is obviously wasteful to use a satellite merely as a point-to-point channel because the satellite is inherently a broadcast medium. A satellite can directly interconnect all users within its range. We will concentrate on the way of using satellites as broadcast channels.

A satellite thus can be considered as a single communication channel that must be shared efficiently and fairly among a large number of dispersed, uncoordinated users. How to share the channel is the issue of this chapter. Before describing satellite packet broadcasting in detail, it is worth mentioning some of the advantages it has over conventional store-and-forward networks that use terrestrial cables.

(i) The protocols are simpler because acknowledgements might not be needed.

(ii) The routing problem vanishes.

(iii) It is no longer possible for some lines to be badly congested while others are idle.

(iv) The topology optimisation problem is reduced to adjusting the satellite bandwidth.

(v) Mobile users can be easily accommodated.

4.3. Conventional Channel Allocation Protocols

4.3.1. Frequency-division multiplexing (FDM)

The traditional way of using a satellite is FDM. If there are N users, the bandwidth is divided up into N equal sized portions, each user being assigned one portion. Since each user has his own private frequency to use, there is no interference between users. FDM is thus a simple and efficient allocation mechanism when there are only a small and fixed number of ground stations, each of which has a steady load of traffic. However, when the number of stations is large and continuously varying, FDM presents some problems. At any instant if fewer than N users use the channel a large portion of the spectrum will be wasted. If more than N users want to use the channel some of them cannot be accommodated for lack of bandwidth.

In most computer systems data traffic is extremely bursty. Consequently, FDM is very inefficient because most of the channel will be idle most of the time.

4.3.2. Fixed Assignment Time Division Multiple Access (TDMA)

In this protocol channel time is administratively assigned to each station according to its

anticipated traffic requirement. the simplest fixed TDMA scheme is illustrated in figure 4.4.1.1.

The frame time T is large enough to contain one time slot for each station. The time slots assigned to each station can be of different durations.

In this protocol all stations constantly monitor the satellite downlink, checking each received packet to see if it addressed to them. If it is, a station delivers it to its user destination, otherwise it ignores it. A more detailed view of how channel time is used is given in figure 4.2.

When data packets are short, several of them can be lumped together in a single burst for more efficient use of the time slot (figure 4.3).

The drawbacks in fixed assignment TDMA are similar to those appearing in FDM:

- fixed assignment does not allow for changes in traffic flow among the stations relative to that assumed when the system is designed.
- fixed assignment performance tends to become poor as station traffic becomes bursty and the number of stations becomes large.

Because of these inefficiencies of the traditional data communication methods several alternative protocols have evolved for packet systems. The next section describes random access protocols.

4.4. Random-Access Protocols

4.4.1. Pure Aloha

This approach was first employed by Abramson at the University of Hawaii. The basic idea of an Aloha system is simple: just let the users transmit whenever they have data to be sent. If no other user is transmitting during this time (and no local noise errors occur at the receiver), the packet will be received successfully by the destination station. If one or more other users are transmitting, a collision occurs and the colliding packets will be destroyed. However, due to the feedback property of packet broadcasting, the sender of a packet can always detect the collision one round-trip time later. If the packet was destroyed the sender just waits a random amount of time and sends it again. The waiting time before retransmission must be random to minimise the probability of further collisions. The described scheme is known as *contention.*

Figure 4.4 shows the generation of packets in an Aloha system.

The channel activity is shown in figure 4.5.

4.4.1.1. Performance

Assume k data sources are each transmitting independently at a Poisson rate of λ messages/sec. Messages are assumed to be of constant length, each τ seconds long.

The maximum possible throughput of such a channel, obtained only if one user were allowed to access the channel and transmit packets continuously with no gaps in the transmission, would be

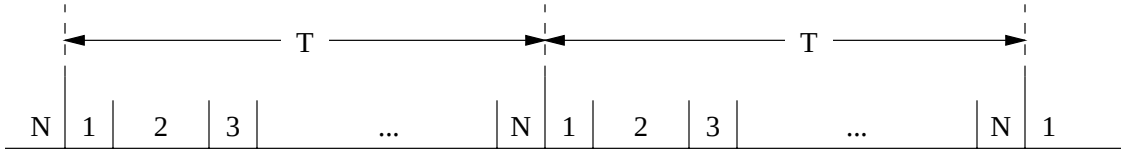


Figure 4.4.1.1.

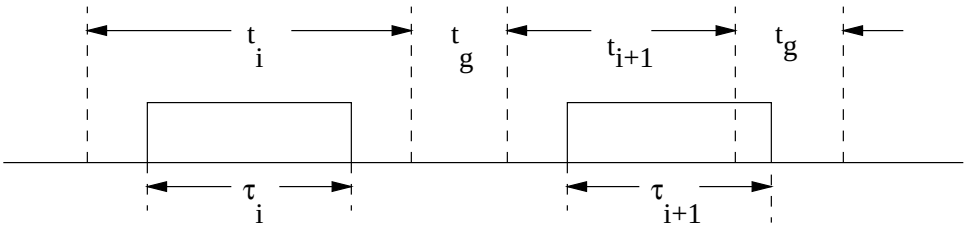


Figure 4.2. Slot, burst, guard times

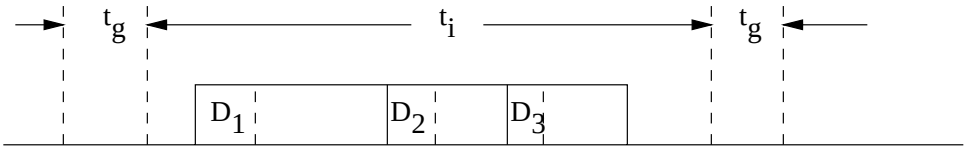


Figure 4.3. Multiple packets per burst

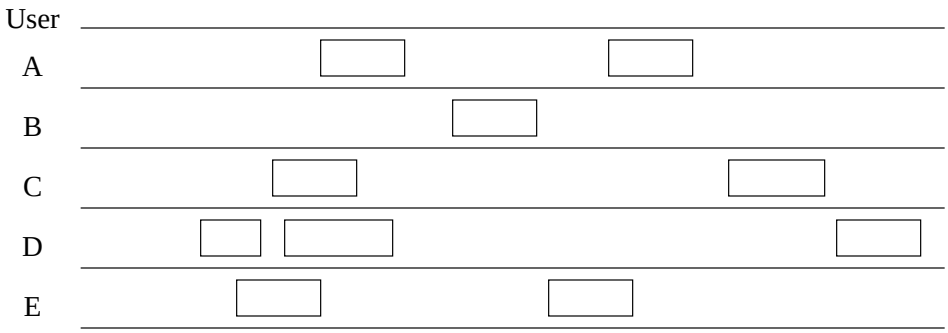


Figure 4.4. Packets are transmitted at completely arbitrary times

$\frac{1}{\tau}$ packets/sec. This is thus the capacity of the channel, C.
With k users the effective utilisation of the channel is

$$S = \frac{k\lambda}{C} = k\lambda t < 1$$

The parameter S thus plays the role of the traffic intensity or utilisation parameter ρ used previously. We shall call S the channel throughput, measured in average number of packets transmitted per packet time.

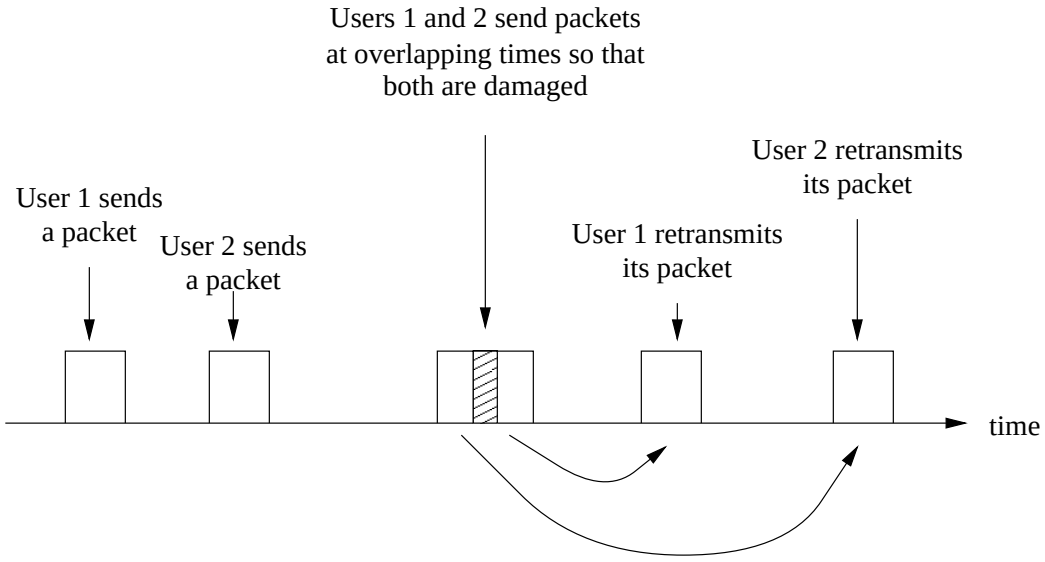


Figure 4.5.

In the Aloha system there will be more than S packets per packet time using the channel because some packets have to be transmitted more than once due to collisions. Let us assume further that the total channel traffic, consisting of newly generated packets plus retransmitted packets, obeys *Poisson distribution.* That is, the probability that n packets are generated during a given p-packet time is given by

$$P_p[n] = \frac{(Gp)^n e^{-Gp}}{n!}$$

A packet will not suffer a collision if no other packets are sent within one packet time of its start, as shown in figure 4.6.

In other words, there is a time period of duration $2/C$ in which no other packet must originate if a collision is to be avoided. From the equation above, the probability that no collision occurs during this 2 packet time is

$$P_2[n = 0] = e^{-2G}$$

and the probability that at least one collision will take place is $1 - e^{-2G}$. The average number of retransmissions is

$$R = G(1 - e^{-2g})$$

and thus

$$G = S + G(1 - e^{-2G})$$

Hence we have

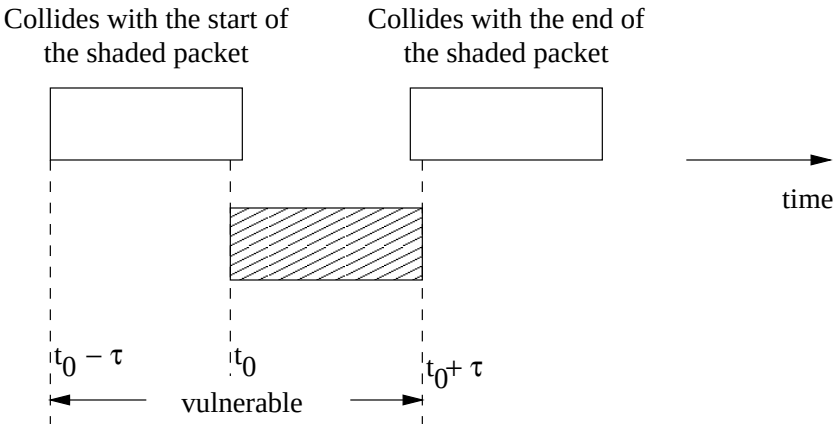


Figure 4.6. Vulnerable for the shaded packet

$$S = Ge^{-2G}$$

This result was first derived by Abramsom (1970). The throughput-offered traffic relation is shown in figure 4.7.

Exercise: Show that the maximum throughput occurs at $G = 0.5$ with $S = \frac{1}{2e} = 0.184$.

Notes:

(a) When variable-length packets are sent, the maximum achievable channel efficiency for the pure Aloha has been shown [Ferguson] to be somewhat less than $\frac{1}{2e}$.

(b) The average channel efficiency can exceed $\frac{1}{2e}$ when the transmission rates of the stations are unequal. This result is known as the excess capacity of an Aloha channel [Abramson 1973] and can give an efficiency approaching 1 in the case of a single user with a high packet rate and all other users with a low packet rate. However, the delay encountered by the low-rate users is significantly higher than in the homogenous case.

4.4.2. Slotted Aloha

In 1972, Roberts proposed a method which has come to be known as *slotted Aloha*. Here the channel time is divided up into discrete intervals (slots). Each interval corresponds to one packet. The timing of the slots is determined by a system wide clock. Each transmission control unit must be synchronised to this clock. With slotted Aloha, a packet is not permitted to be sent whenever it is generated. Instead it is required to wait for the beginning of the next slot.

Figure 4.8 shows a slotted Aloha channel. packet B and C on this diagram are in collision.

If a given packet is transmitted beginning at time t_0, then another packet will collide with it if it originates between time $t_0 - \tau$ and t_0. That is, the vulnerable period is now reduced in half and this leads to

$$S = Ge^{-G}$$

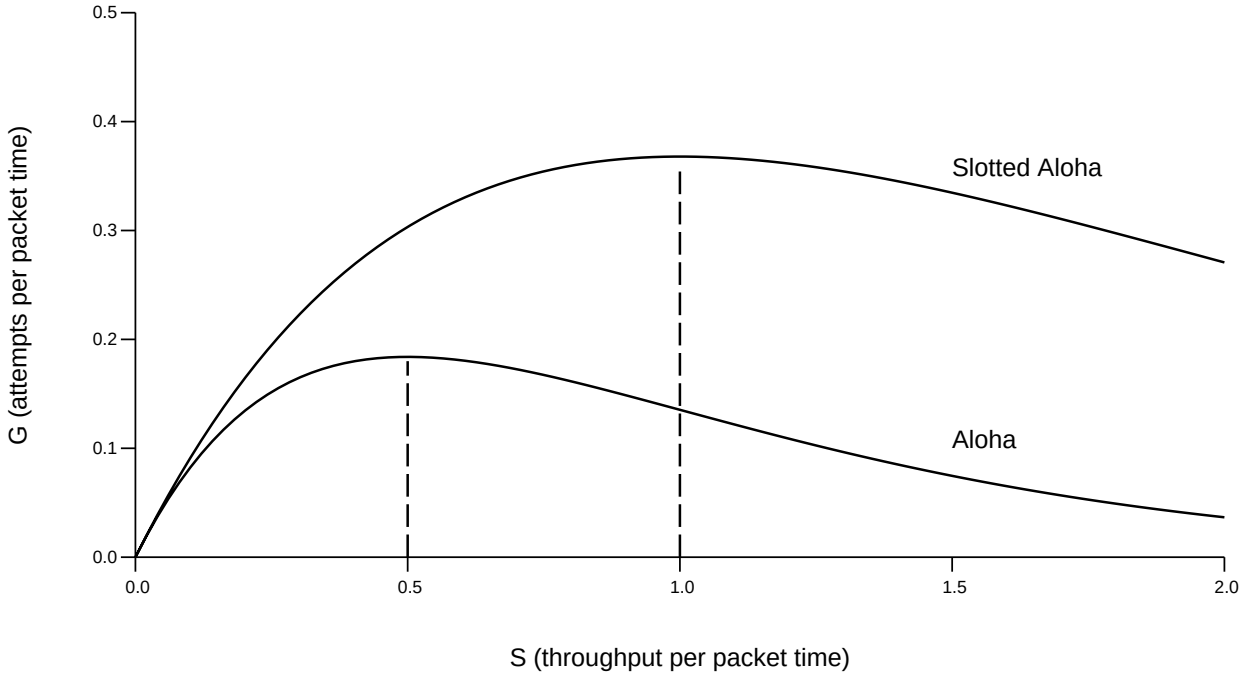


Figure 4.7.

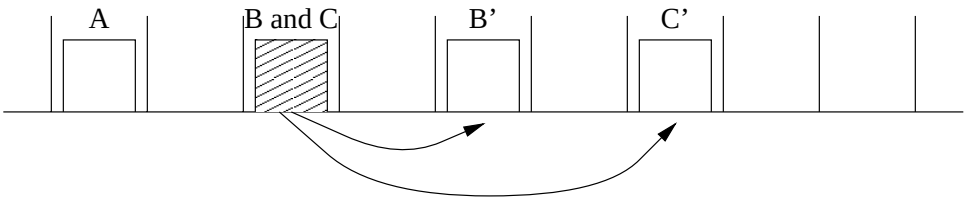


Figure 4.8. A slotted Aloha channel. Each packet must begin on one of the time slots.

It can be shown that the maximum throughput of a slotted Aloha is $\frac{1}{e}$ or 36.8% at $G = 1$ (figure 4.7).

4.4.3. Reservation Aloha

While the excess-capacity effect does allow a small number of high-rate stations to use the channel, it would appear desirable to remove these stations from the random-access competition for channel time. Reservation Aloha methods were designed to do this while maintaining the relative simplicity of slotted Aloha. In these methods some slots are reserved for specific stations. Other stations are restrained from using a reserved slot. These methods differ in the way reservations are made and released.

4.4.3.1. Binder (1975)

This method is basically TDM modified to slotted Aloha for low channel utilisation. As in TDM each of N stations "owns" one slot pre time frame. If there are more slots than stations, the extra slots are not assigned to anyone. If the owner of a slot does not want it during the current frame, he leaves it empty. During the next frame, the slot becomes available to anyone who wants it on

a contention basis. If the owner wants to retreive the slot, he transmits a packet, thus forcing a collision (if there was other traffic). After a collision, everyone except the owner must desist from using the slot. Figure 4.9 shows the Binder reservation scheme in which a frame has 8 slots, 7 of which are owned.

One slight inefficiency with this method is that whenever the owner of a slot has nothing to send, the slot must go idle during the next frame. Also that after each collision, the collidees must abstain for one frame to see if the owner wants the slot back.

4.4.3.2. Crowther (1973)

This reservation scheme is applicable even when the number of stations is unknown and varying dynamically. In this method the frame size is equal to or greater than the maximum uplink plus downlink propagation time, allowing each station to remember the state of a slot's previous usage at the time a new usage of it must begin. Three states are distinguished by each station:

State 1: Empty; the slot was nut successfully used by anyone. During this frame everyone can compete for it on a contention basis.

State 2: Other; the slot was used by another station. Do not use it.

State 3: Mine; the slot was last used by itself. It is reserved for that station's exclusive use.

Thus if a station has a long queue of messages, once it successfully uses a slot it is guaranteed successive use of it until its queue is empty. This is illustrated in figure 4.10.

If traffic is equally distributed among the stations and duty cycles are high, the system behaviour approaches that of fixed TDMA. If duty cycles are low, it approaches simple slotted Aloha behaviour [Lam, 1978].

If a significant traffic imbalance exists even for a short period of time, however, it is possible for a single station to capture the entire channel, creating long delays for other stations. Other issues concern whether a station should wait for its acquired slots or also use intervening empty slots. Because of these issues we will consider explicit reservation schemes in the next subsection.

4.5. Explicit Reservation Protocols

The basic approach consists of transmitting a relatively small number of bits to request channel time for queued messages, which are subsequently sent at the requested time.

4.5.1. Roberts Reservation Scheme

In this scheme, each frame consists of M equal-sized data slots, followed by a special slot, which is divided into V smaller subslots used to make reservations.

If a station wants to send data, it broadcasts a short request packet (using the slotted Aloha protocol) during one of the reservation subslots. If the reservation is successful (i.e. there is no collision), then the next data slot (or slots) is reserved. At all times every station monitors all channel traffic and must keep track of the queue length J, so that when any station makes a

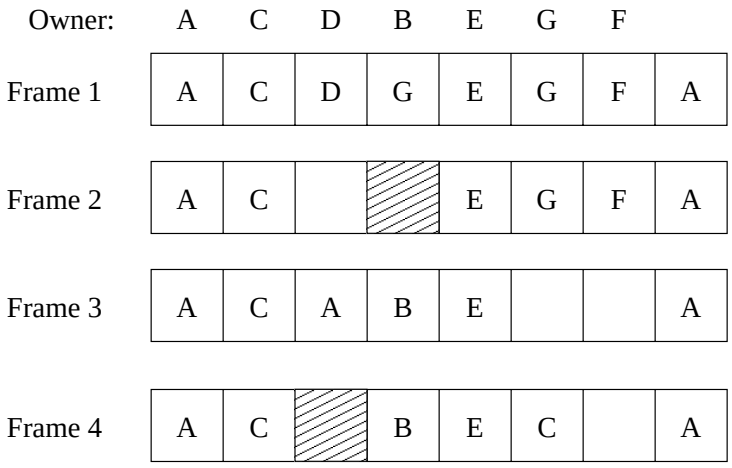


Figure 4.9.

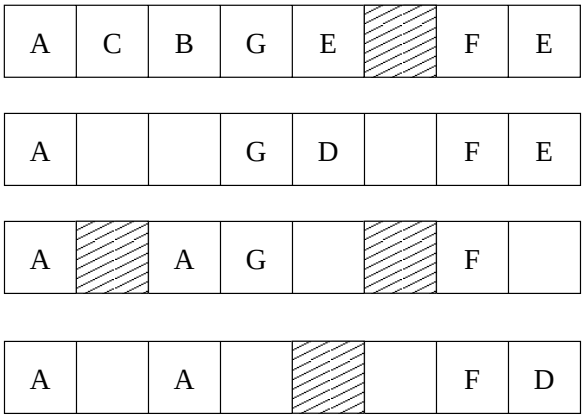


Figure 4.10.

successful reservation it will know how many data slots to skip before transmitting.

When the queue length drops to zero, all slots revert to reservation subslots to speed up the reservation process. Robert's reservation system is illustrated in figure 4.11.

4.5.2. Reservation-TDMA

Reservation-TDMA (R-TDMA) uses fixed assignments to make reservation. Channel time is divided as shown in figure 4.12, with N small reservation slots of duration τ' and $k \times N$ large slots of duration τ in each frame.

One reservation slot in each frame is preassigned to each of the N stations. Each station is also preassigned one data slot in each of the k subgroups and is called the "owner" of the slot.

Reservations are made as follows: During its reservation slot each station sends a "new reservation count" to announce whether any new packets have arrived since its last reservation slot. Every station monitors its channel for these reservation packets and, at a globally agreed-upon time (the update slot), adds the received values to a reservation table which is used to assign the data slots.

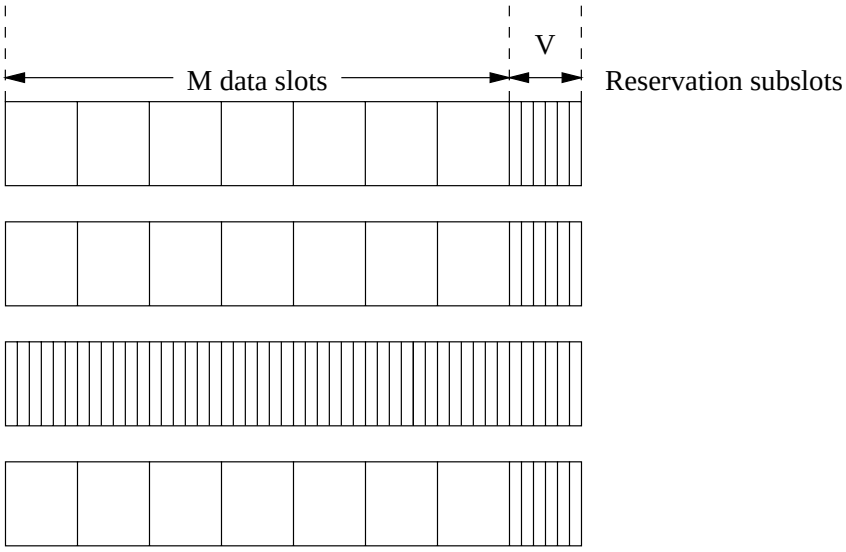


Figure 4.11.

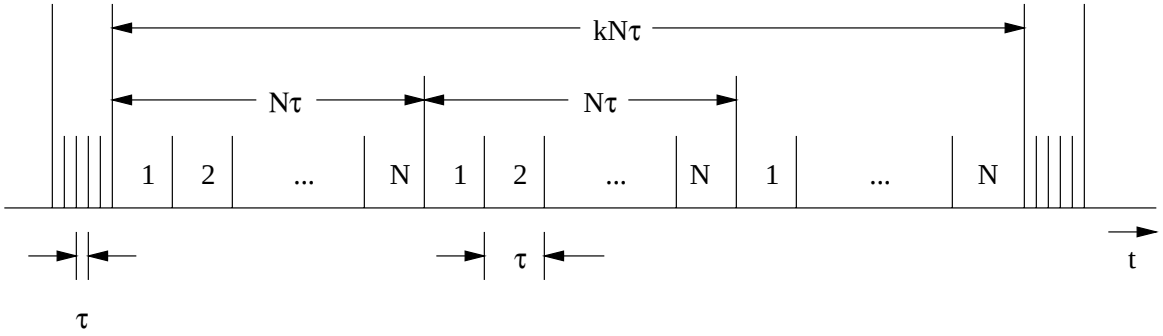


Figure 4.12.

If a slot's owner has one or more reservations in the table, the slot is assigned to the owner. Slots whose owners have a zero table entry are assigned in a round-robin manner among those stations with non-zero entries. Each time an assignment is made, the assigned station's table entry is decremented. If the reservation table is empty, a station is allowed to send immediately in its own slot.

Chapter 5. Local Area Networks

5.1. Ethernet

5.1.1. Brief Description

Ethernet is a system for local communication among computing stations. The Ethernet uses tapped coaxial cables to carry variable length digital data packets among various stations or devices. The transmission cable (called Ether) is passive and shared by all stations. A station interfaces to the Ethernet through a transceiver which is tapped into the cable using off-the-shelf CATV (cable television) taps.

A source station broadcasts its packet onto the Ether. The packet is heard by all stations, however, it is copied from the Ether only by the destination station which is identified by the packet's leading address bits. Access to the coaxial cable (Ether) is based on a modified contention scheme. Figure 5.1 shows a two-segment Ethernet.

5.1.2. Mechanism

Packet broadcasting was discussed in the previous chapter. The access method to the transmission medium is on a contention basis. In Ethernet the access method is modified to what is called CSMA/CD (carrier sense multiple access with collision detection). The Ethernet mechanism is modelled in Figure 5.2.

5.1.3. Carrier Detection

As a packet's bits are placed on the coaxial cable by a station, they are phase encoded in such a way that there is at least one transition during each bit time (see Figure 5.3). The passing of a packet on the cable can therefore be detected by listening for its transitions. This is referred to as "detecting the presence of carrier".

The ALOHA network does not have carrier detection and consequently suffers a substantially higher collision rate.

5.1.4. Contention algorithm

When a packet is going by, a station interface can hear its carrier and so does not initiate a transmission of its own.

When a carrier is absent a station may decide to transmit its data. However, it takes about 4μs for an electrical signal to pass from one end of a 1km coaxial cable to the other end. During this 4μs more than one station might decide to transmit data simultaneously, not knowing each other's carrier already existed on the cable. This results in a collision.

The transceiver must listen to the cable when it is transmitting. If it notices a difference between

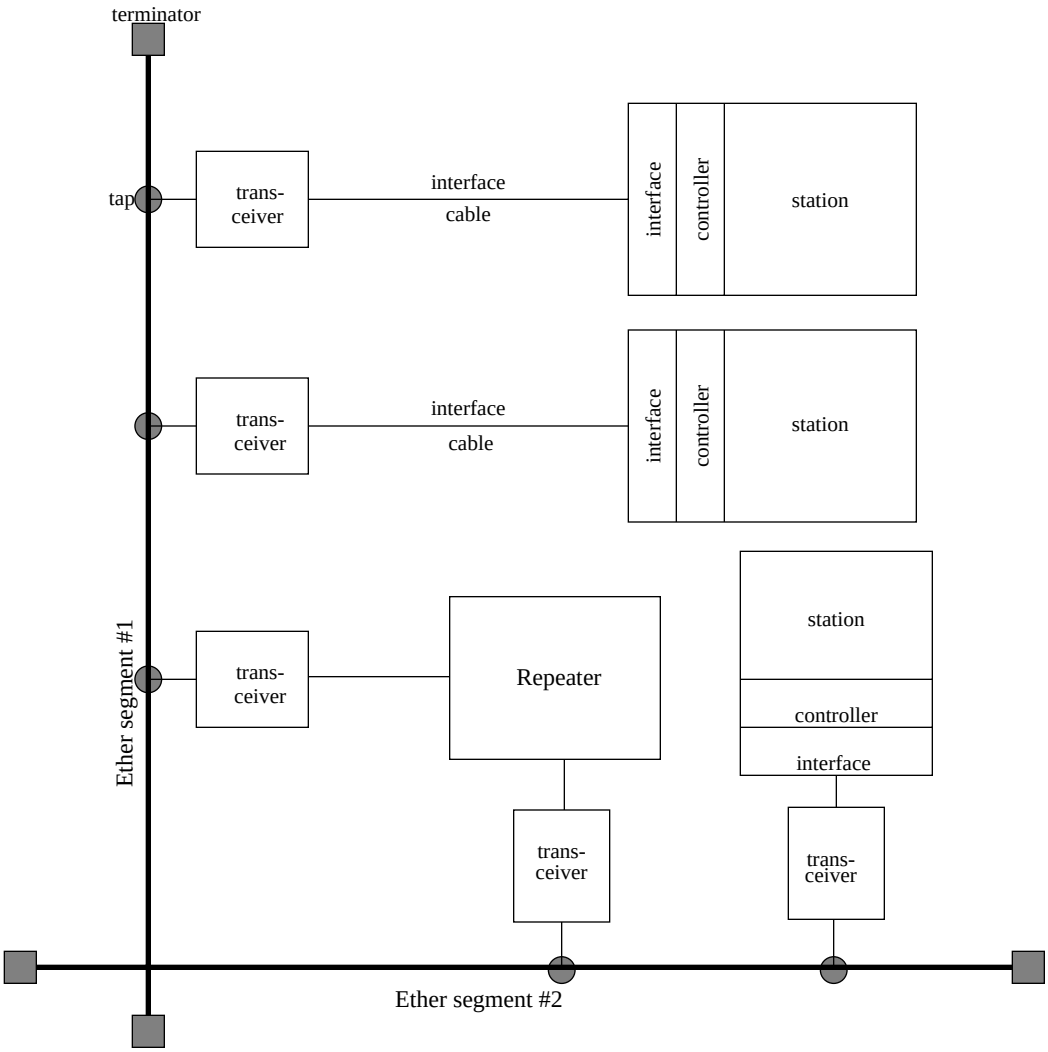


Figure 5.1. A two-segment Ethernet

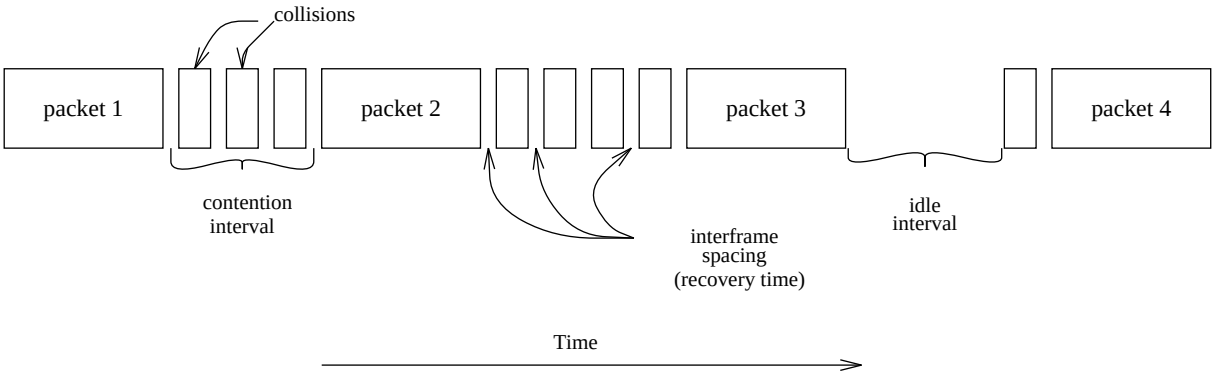


Figure 5.2. Contention based access allows any station to use the bus at any time provided it first checks to see that the bus is not in use. Collisions occur when two stations try to start at the same time.

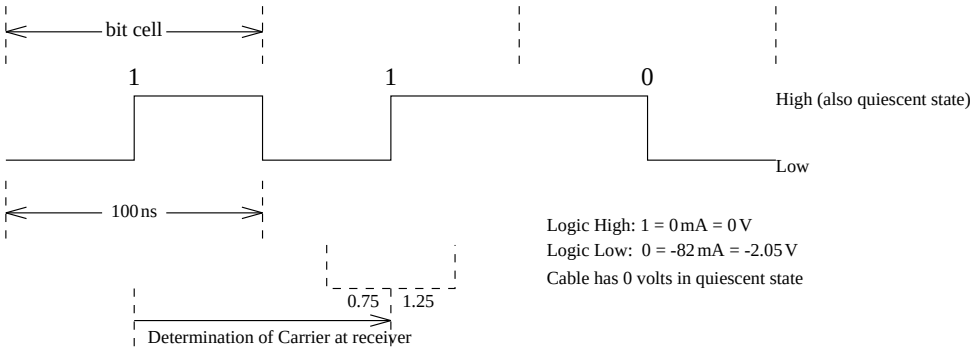


Figure 5.3. Channel Encoding: Manchester encoding is used on the coaxial cable. It has a 50% duty cycle, and ensures a transition in the middle of every bit cell. The first half of the bit cell contains the complement of the bit value and the second half contains the true value of the bit.

what it is transmitting and what it is receiving, it knows that a collision is taking place. Each station thus detects the collision, aborts its transmission, waits a random period of time, and then tries again. The random waiting time is necessary to minimise the chance of another collision. The model for Ethernet will therefore consist of alternating contention and transmission periods, with idle periods occurring when all stations are quiet (Figure 5.2).

5.1.5. Binary exponential backoff

Binary exponential backoff is a waiting time randomisation strategy used by Metcalfe and Boggs to minimise the transmission delay under light loads and yet be stable under heavy loads. The strategy proceeds as follows. If there is a collision, all colliding stations set a local parameter, L, to 2 and choose one of the next L slots for retransmission. Every time a station is involved in a collision, it doubles its value of L. Thus after k collisions, the chance of another collision in each of the succeeding slots will be $1/2^k$. Figure 5.4 shows three stations involved in a collision and their retransmissions.

5.2. Topology and Packet Formats

5.2.1. Topology

The Ethernet uses a simple unrooted tree topology. Since it is a tree there is exactly one path between any pair of stations and thus the network does not suffer from multipath interference. The Ether can branch to any convenient place and because it is unrooted it can be extended from any of its points in any direction. Any station wishing to join an Ethernet taps into the Ether at the nearest convenient point.

5.2.2. Packet format

The format of the packet is simple as shown in Figure 5.5. It consists of an integral number of 16-bit words. The first word contains the source and destination addresses - each of 8 bits. The last word contains a 16-bit cyclic redundancy checksum for end-to-end detection of transmission errors.

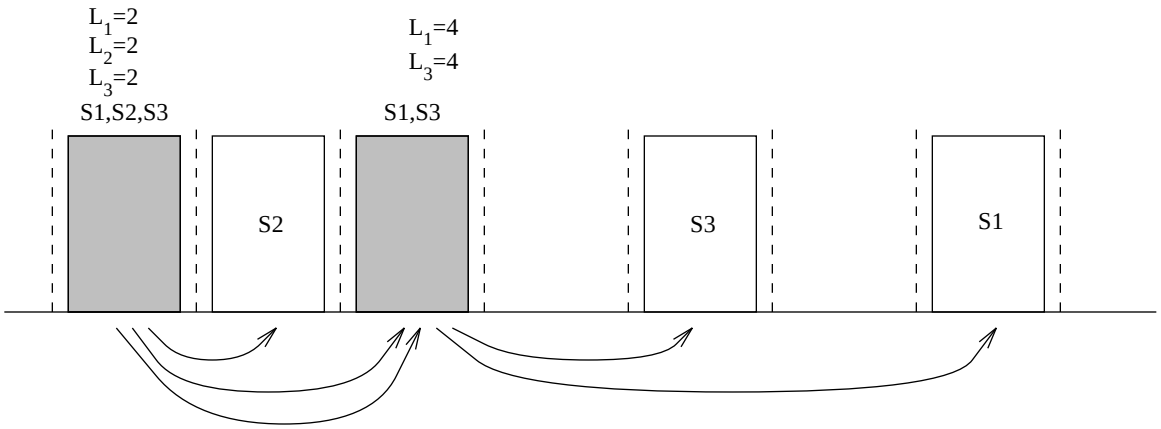


Figure 5.4.

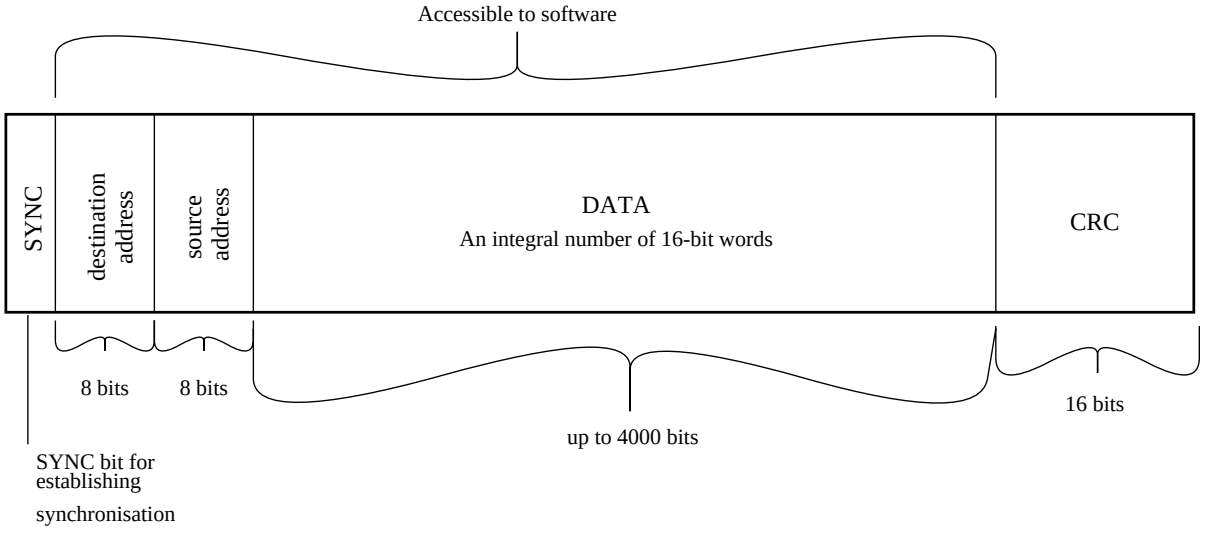


Figure 5.5. The Ethernet packet format

A SYNC bit precedes each packet transmitted. A receiving interface uses the appearance of carrier to detect the start of a packet and uses the SYNC bit to acquire bit phase. The receiving interface knows when a packet ends because the carrier ceases. It then checks that an integral number of 16-bit words has been received and that the CRC is correct. The last word received is assumed to be the CRC and is not copied into the packet buffer.

When the hardware detects the start of a packet it checks its destination address. If that does not match its own address it ignores the packet. By convention, a packet with a destination address of zero is called a *broadcast packet* and is intended for all station.

5.2.2.1. Performance

The simple model of the Ethernet we use to look at its performance is shown in Figure 5.1. We will examine the performance under conditions of heavy and constant load, that is, there will be no idle intervals. In the worst case it takes a station a period of $2T$ to detect a collision after

starting a transmission, and for this reason we will model the contention interval as a slotted ALOHA system with slot width $2T$.

Let P be the number of bits in an Ethernet packet and C be the peak capacity in bits per second carried on the Ether.

Now under the stated conditions we assume that there are k stations always ready to transmit. We also assume that a queued station attempts to transmit in the current contention slot with probability p, or delays with probability $1-p$.

5.2.2.2. Acquisition Probability

The probability A that exactly one station attempts a transmission in a slot and therefore acquires the Ether is

$$A = kp(1-p)^{k-1}$$

(show that A is maximised when $p = 1/k$).

5.2.2.3. Waiting Time

Let W be the mean number of slots of waiting in a contention interval before a successful acquisition of the Ether by a station's transmission.

The probability of waiting no time at all is just A.

The probability of waiting one slot is $A(1-A)$.

The probability of waiting $i$slots is $A(1-A)^i$.

So the mean W is shown to be

$$W = \frac{1-A}{A}$$

Exercise

Show that $W = \dfrac{1-A}{A}$

$$W = \sum_{i=0}^{\infty} iA(1-A)^i$$

5.2.2.4. Efficiency

The channel efficiency E is defined as the fraction of time that the Ether is carrying good packets. The Ether's time is divided between transmission intervals and contention intervals.

A packet transmission takes P/C seconds. The mean time to acquisition is $W \times 2T$. Therefore, by our simple model,

$$E = \frac{P/C}{P/C + 2WT}$$

Exercises

1. If the Ethernet is constantly under very heavy load conditions, i.e. $k - > \infty$, find the worst value of W and the channel efficiency.

2. Draw the channel efficiency as a function of the number of stations queued. For the cases

   (a) the packet size P = 1024 bits

   (b) P = 256 bits

   (c) P = 128 bits

   Assuming that the data rate is 10Mbps, the cable length is 1km and the signal propagates with velocity 200000km/s.

5.3. Ring Networks

Another popular kind of local network is the *ring network*. Ring nets also use cables, twisted pair, coaxial or fibre optic, just like carrier sense networks, but the organisation is fundamentally different.

The access method employed in ring networks is called token-passing. Token-passing is essentially a time-sharing scheme. It permits a station to transmit only at stated times, when it possesses a "token", allowing the station to use the network (figure 5.6). When a node completes a transmission or when an allotted maximum time elapses, it transmits the token to the next node in a prescribed sequence.

We will discuss a *few major types* of ring networks in the following subsections.

5.3.1. Token Ring

The token is a special 8-bit pattern, say 11111111. In this kind of ring the token circulates around the ring whenever all stations are idle. When a station wants to transmit a packet, it has to seize the token and remove it before transmitting.

To remove the token, the ring interface, which connects the station to the ring (see figure 5.7), must monitor all bits that pass by. When the last bit of the token passes by, the ring interface inverts it, changing the token into another bit pattern, known as a *connector* (11111110). The station which so transformed the token is immediately permitted to transmit its packet.

This mechanism has several important implications for the ring design:

(a) The interface must be able to read and store the last bit of a potential token, and decide whether or not to invert it before forwarding it to the ring. This creates a 1-bit delay in each

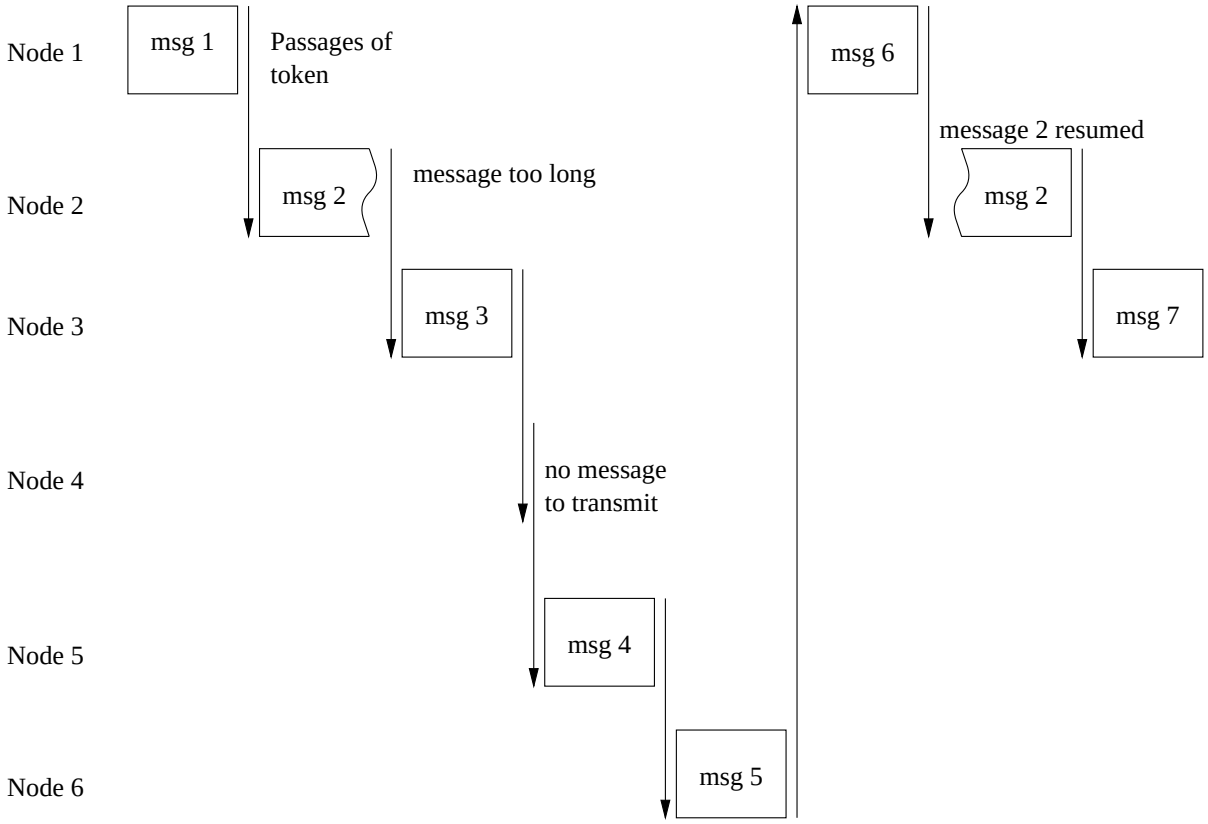


Figure 5.6.

ring interface.

(b) The ring must be long enough (or artificial delays may have to be inserted) to contain a complete token to circulate when all stations are idle.

5.3.1.1. Operating Modes

Ring interfaces have two operating modes, listen and transmit, as shown in figure 5.8.

In the listen mode, the input bits are simply copied to output, with a delay of one bit time (figure 5.8(a)).

The transmit mode is entered only after the token has been converted to the connector. The interface breaks the connection between input and output, and enters its own data onto the ring.

As soon as a station finishes transmitting the last bit of its packet it must regenerate the token. When the last bit of the packet has gone around and come back, it must be removed, and the interface must switch back to listen mode immediately, to avoid removing the token or connector that follows.

To make sure that the packet is correctly received by the intended destination an acknowledgement is required. The packet format includes a 1-bit field after the checksum

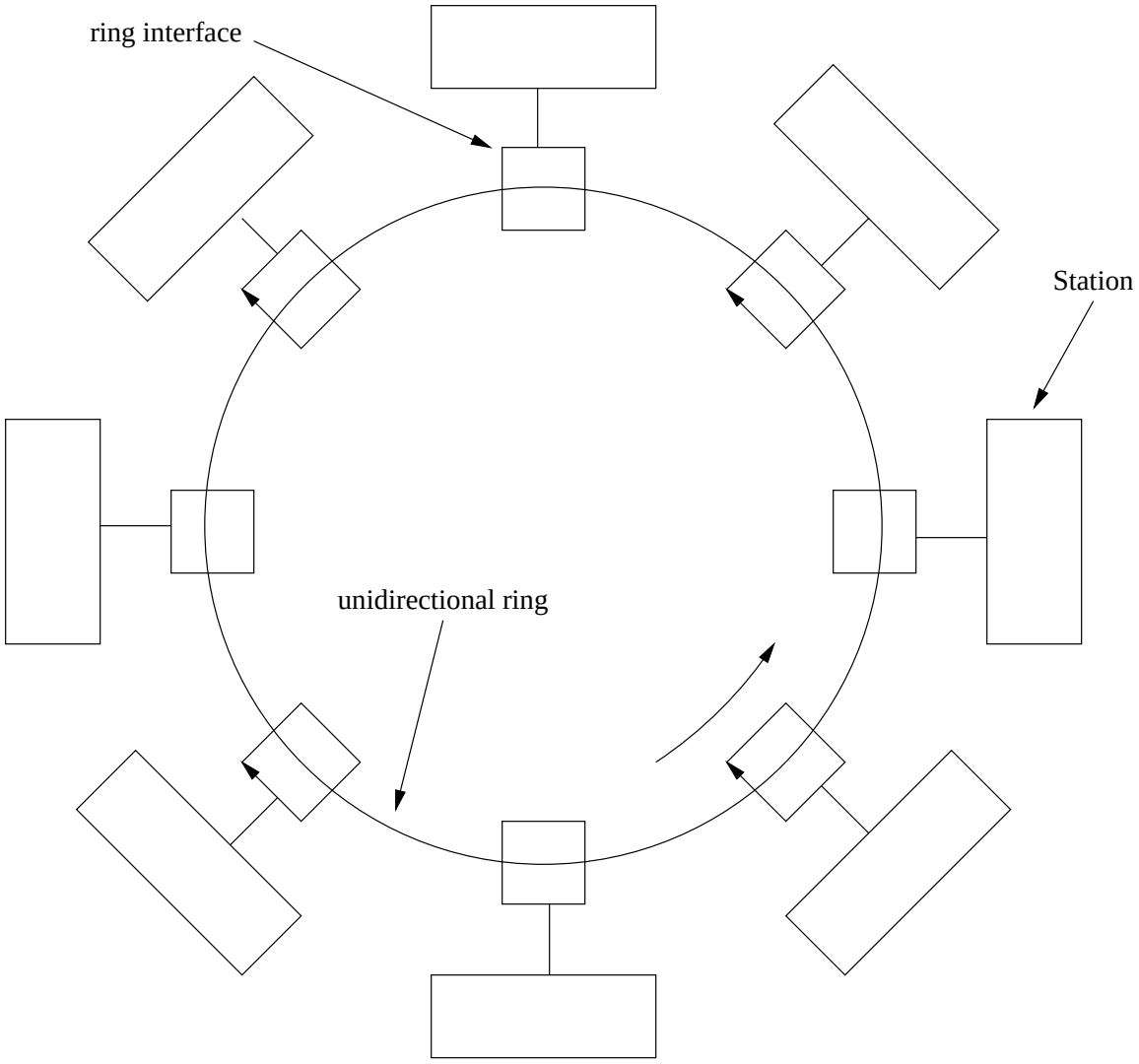
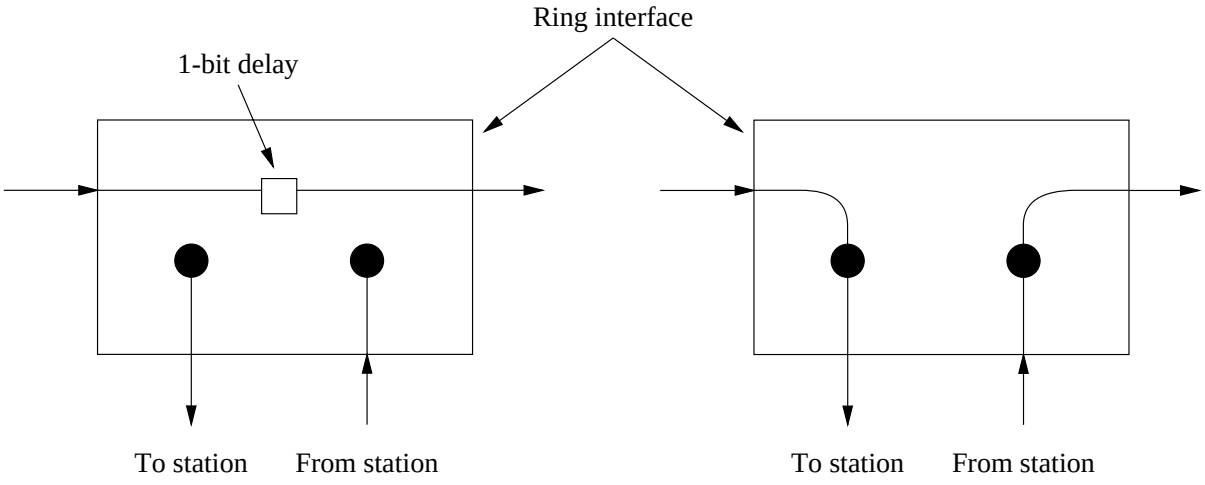


Figure 5.7.



(a) Listen mode

(b) transmit mode

Figure 5.8.

for acknowledgements. This bit is initially cleared (i.e. zero). When the destination station has received a packet, and the checksum has been verified correctly the acknowledgement bit is set to 1.

5.3.1.2. Performance

When traffic is light, the token will spend most of its time idly circulating around the ring. When the traffic is heavy, as soon as a station finishes its transmission and regenerates the token, the next station inline will seize the token and transmit its queued packets. In this manner the ring nets are similar to systems that use hub polling. In both cases the control token is passed smoothly in round-robin fashion from station to station. The major difference is that with conventional hub polling the master station initiates the poll, whereas in the ring net all stations are equals.

To analyse the performance of a token ring we assume:

- Packets are generated according to a Poisson process, the total arrival rate of all N stations combined is λ packets/sec.
- The service rate is μ packets/sec; this is the number of packets/sec that a station can transmit.
- When a station is permitted to send it sends all its queued packets, with the mean queue length being q packets.

The quantity we are interested in is the "scan time", s, the mean interval between token arrivals at a given station. The scan time is the sum of the walk time w (propagation delay and 1-bit delays at the interfaces around the ring) and the service time required for Nq queued packets, i.e.

$$s = w + \frac{Nq}{\mu}$$

The mean queue length q at each station is

$$q = \frac{\lambda s}{N}$$

Hence

$$s = w + \frac{\lambda s}{\mu}$$

Introducing $\rho = \frac{\lambda}{\mu}$ and solving for s we find

$$s = \frac{w}{1-\rho}$$

The channel-acquisition delay is about half the scan time. Thus from the last equation we notice that

- The delay is always proportional to the walk time, and hence to the length of the ring, both for low and high load.
- The delay increases rapidly when the utilisation factor of the entire ring ρ tends to 1.

The token ring works fine as long as all the interfaces function flawlessly. However the system becomes useless when the token is accidentally lost. One way to get around this problem is to have a timer in each of the interfaces. the timer is set just before the interface searches for a token. If the timer goes off the interface knows that the token is lost and it then puts a new token onto the ring. However, if two or more interfaces decide to put tokens onto the ring at the same time the system is in real deep trouble.

5.3.2. Contention Rings

As the name implies, in a *contention ring*, a station utilises the contention mechanism to acquire the control token when the channel is under light load.

The contention ring contains nothing at all when there is no traffic. When a station has a packet to transmit, it first listens to see if there is any traffic passing by its interface.

- If there is no traffic on the ring it just starts transmission. At the end of its packet, it puts a token on the ring, just as the token ring does. However, if the token remains uncaptured after travelling around the ring the interface removes the token and the ring becomes quiet once again.
- If there is traffic on the ring the interface waits for the token, converts it to a connector and then transmits its own packet. When it finishes the transmission it generates the token to the ring. When the traffic is heavy another station will capture the token and repeat the process. Consequently, under conditions of high steady load the contention ring acts just like a token ring.

The trouble occurs when two or more stations believe that the ring is idle and simultaneously decide to transmit packets, and this results in a collision. A station detects a collision by comparing what it is transmitting and what it is receiving. When it detects a collision, it terminates its transmission but keeps on absorbing data until there is no more. Eventually the ring becomes quiet, the colliding stations wait a random amount of time and then try again.

In brief, the contention ring has low channel-acquisition delay when it is lightly loaded and the properties of a token ring when it is heavily loaded.

5.3.3. Register Insertion Rings

Figure 5.9 shows a ring interface of the register insertion ring. The interface contains two registers, a shift register for receiving packets destined for this station or passing through this station and an output buffer for holding packets to be sent by this station. The maximum size of a packet is obviously the length of the output buffer.

When the ring is first started up, the input pointer points to the extreme right bit position in the shift register, i.e. the register is empty. When data arrive from the ring bit by bit they are placed in the register from right to left, the pointer moving along as well. As soon as all the bits of the destination address field have arrived the interface works out whether the incoming packet is destined for it or not. If it is, the packet is removed from the ring into the station. The pointer is then reset to the extreme right position.

If, however, the packet is not addressed to the local station, the interface forwards it onto the ring. As each new bit arrives, it is put in the place pointed to by the input pointer. The entire contents of the shift register is then shifted right one position, pushing the rightmost bit out onto the ring. The input pointer is not advanced. If no new input arrives the contents of the shift register can be reduced by one bit and the input pointer moves right one position.

Whenever the shift register has pushed out the last bit of a packet, it checks to see if

(i) there is an output packet waiting, and

(ii) the number of empty slots in the shift register is at least as large as the output packet.

Only if both conditions are met can the output proceed, in which case the output switch is flipped and the output buffer is now shifted out onto the ring, one bit at a time, in synchronisation with the input. New input is accumulated in the shift register which is why there must be enough empty slots there to accommodate all the intput while the output buffer is being emptied. As soon as the output buffer has been emptied, the switch is flipped back again to the shift register.

5.4. Token Passing versus CSMA/CD

The method of accessing the network by any node (station) strongly influences the choices of network topology. Bus networks like Ethernet and Z-net employ CSMA/CD access, while ring networks like Domain employ token passing. We will briefly discuss the main and contrasting points of the two access schemes.

(a) CSMA/CD is a random scheme in which a node accesses the bus on a contention basis. This scheme offers two major advantages:

  (i) extreme simplicity

  (ii) suitability for integration into inexpensive LSI chips.

  It works well for lightly loaded systems, yielding better than 90% channel capacity utilisation and low access delay.

  For heavily loaded systems, CSMA/CD has some serious drawbacks.

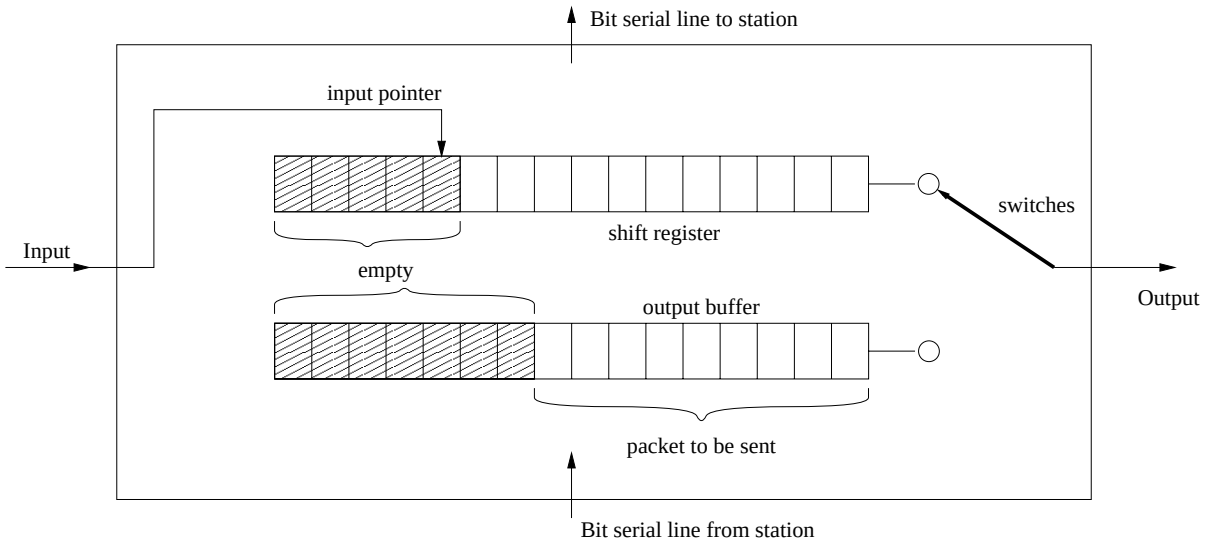


Figure 5.9. Register insertion ring

(i) The greater the channel traffic the greater the probability of messages colliding. This leads to unpredictable transmission delays. For this reason the access method is not strictly suitable for real-time process controls without modification.

(ii) Another drawback is the lack of a built-in prioroty mechanism. Messages with different urgency levels receive the same treatment when contending for the bus.

(b) In token passing type LANs, a message token is passed from station to station allowing each station to transmit packets in an orderly manner. this type of access approach is highly deterministic, and its message-transmission times are predictable. Furthermore, the fact that the sender is also the receiver for acknowledgement in a ring makes this type of access highly reliable and thus suitable for applications like process control. In addition, token passing is not constrained by packet size or data rate; it works just as well under varying message size and data rate conditions. Token passing access is not confined to a ring architecture; a token is passed onto each node in much the same way as is done in a ring network. However, token passing also has some disadvantages.

(i) The interfaces used on a ring are active rather than passive. Any break in the ring, whether at a station or in the cable, can bring down the entire network.

(ii) An access control token could be destroyed or even mistaken and it is a difficult task to recover.

(iii) Token passing is more complex than CSMA/CD and this makes the integration into LSI ICs much more difficult.

5.5. Broadband versus Baseband

[Note: I have not changed this section from the original written in 1982.]

Multiconductor, twisted-pair, coaxial and fibre optic cable are all suitable for local area networks. However, baseband or broadband coaxial cable and fibre optic cable are most often selected to handle these transmissions.

Baseband coax is more economical. Unlike broadband cable systems, baseband cable systems do not require data modulation onto an rf carrier. Consequently, expensive items like modems are not required. Baseband systems appear ideal for application involving low data rates (less than 3 Mbps) over short cable lengths (less than 2 km). Baseband coaxial cables are being successfully implemented in such networks as Ethernet, Z-net, Primernet, HYPERchannel and Net/one.

For higher data rates and longer distances, broadband coaxial cable is more practical. Furthermore it is predictable that future LANs must accommodate both voice, video and digital data traffic and boradband coax as well as fibre-optic cables are the choices.

However, for fibre optic, they are still not cost effective for the multidrop applications in local networks. Fibre optic cables are capable of carrying very high data rates (Gigabits/sec) and this makes them suitable for long distance communications between networks, for environments where electrical noise isolation and electromagnetic interference are critical.

Chapter 6. Flow Control

6.1. Introduction

A packet-switched network may be thought of as a pool of resources (channels, buffers and switch processors) whose capacity must be shared dynamically by many users wishing to communicate with each other. Each user must acquire a subset of the resources to perform its task (transfer data from source to destination). If the competition for the resources is unrestricted and uncontrolled, we may encounter problems such as loss of efficiency, unfairness, congestion and deadlock.

6.2. Flow Control: problems and Approaches

In this section, we first describe the congestion problems caused by lack of control. Then we define the functions of flow control and the different levels at which these functions are implemented.

6.2.1. Problems

6.2.1.1. Loss of Efficiency

Network operation becomes "inefficient" when resources are wasted. This may happen either because conflicting demands by two or more users make the resource unusable, or because a user acquires more resources than strictly needed, thus starving other users.

Example: An example of wastage caused by buffer capture is shown in Figure 6.1.

We have a single network node with two pairs of hosts, (A, A') and (B, B'). The traffic requirement from A to A' is 0.8. The requirement from B to B' is variable and is denoted by λ.

When $\lambda \rightarrow 1$, the output queue from the switch to host B' grows rapidly filling up all buffers in the switch. The total throughput, i.e., the sum of (A, A') and (B, B') delivered traffic as a function of λ is plotted in Figure 6.2.

- For $\lambda < 1$, throughput $= \lambda + 0.8$
- For $\lambda > 1$, throughput drops to 1.1. This is because the switch buffers become full causing built-up queues in both A and B hosts. The probability that a packet from B captures a buffer in the switch is 10 times that from A. Since the (B, B') throughput is limit to 1, the (A, A') throughput is reduced to 0.1, yielding a total throughput 1.1.

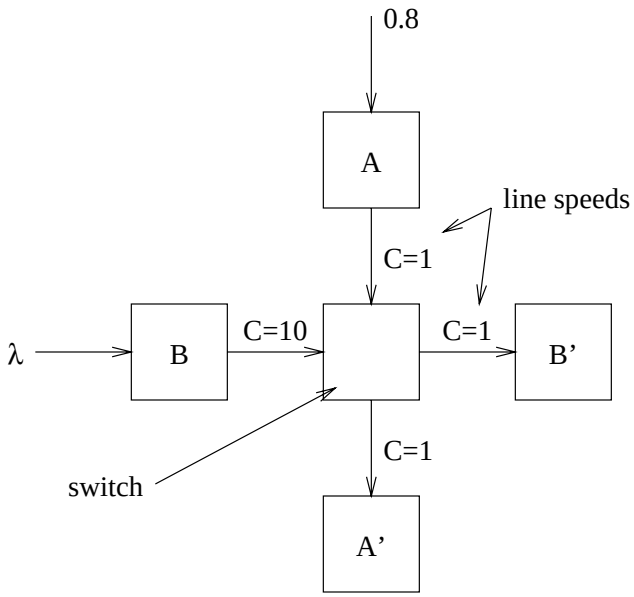


Figure 6.1.

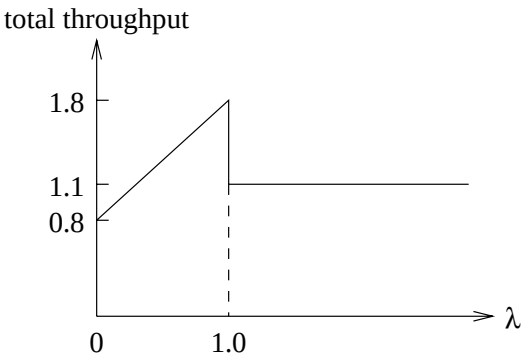


Figure 6.2.

6.2.1.2. Unfairness

This is just a result of uncontrolled competition. Some users might have advantages over others (capture a larger share of resources) either because of their relative position in the network or because of the particular selection of network and traffic parameters.

In the above example *B* obviously enjoys better treatment than *A*. Another example of unfairness is depicted in Figure 6.3.

6.2.1.3. Congestion and Deadlock

Congestion can be defined as a situation when an increase of offered load beyond the critical system capacity causes a decrease in useful throughput (see Figure 6.2).

The ultimate congestion is a deadlock i.e. when the network crashes. Figure 6.4 illustrates a direct store-and-forward deadlock. Two adjacent nodes want to send packets to each other, but each is full and cannot accept them. They are both stuck in this lock up situation.

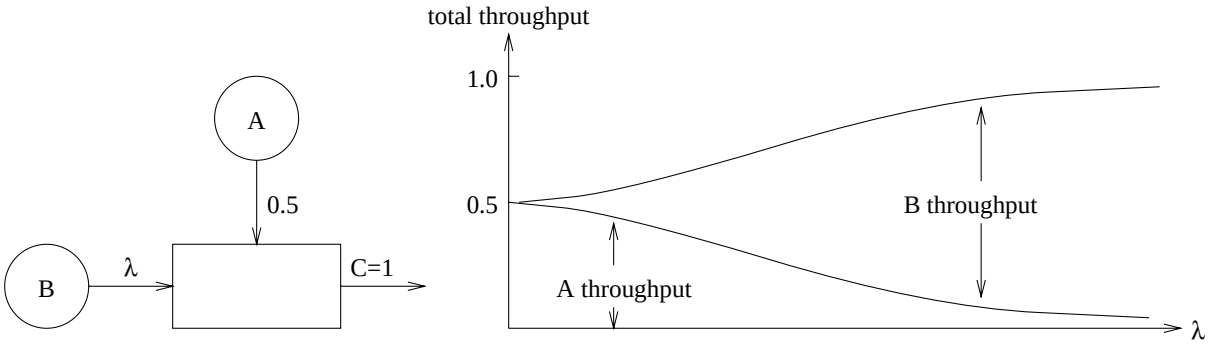


Figure 6.3.

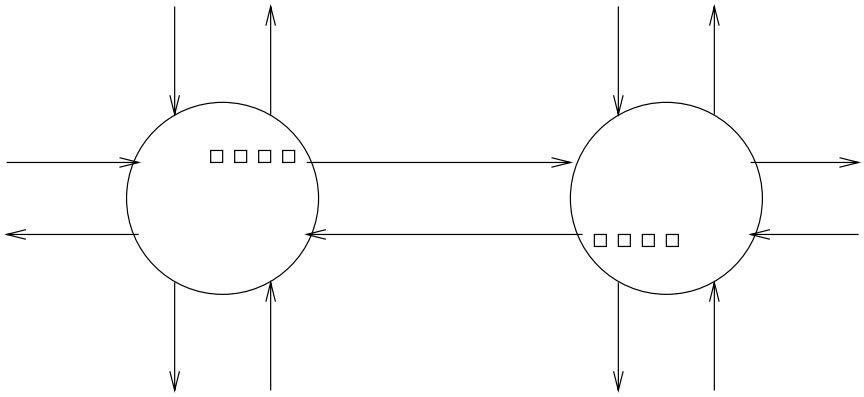


Figure 6.4.

6.2.1.4. Flow Control Functions

Flow control is a set of protocols designed to maintain the flow of data within limits compatible with the amount of available resources. In particular, flow control aims to

- maintain efficient network operations
- guarantee fairness to a certain degree in resource sharing
- protect the network from congestion and deadlock.

These goals are not achieved without a cost. The flowcontrol performance tradeoff is shown in Figure 6.5.

The controlled throughput curve is lower than the ideal curve because of control overhead. The control overhead is due to the exchange of information between nodes and utilization of some resources required for control functions.

Different flow-control procedures may operate at different levels in the network. In the following subsection we will identify various levels of protocols with their associated problems and the appropriate flow control protocols.

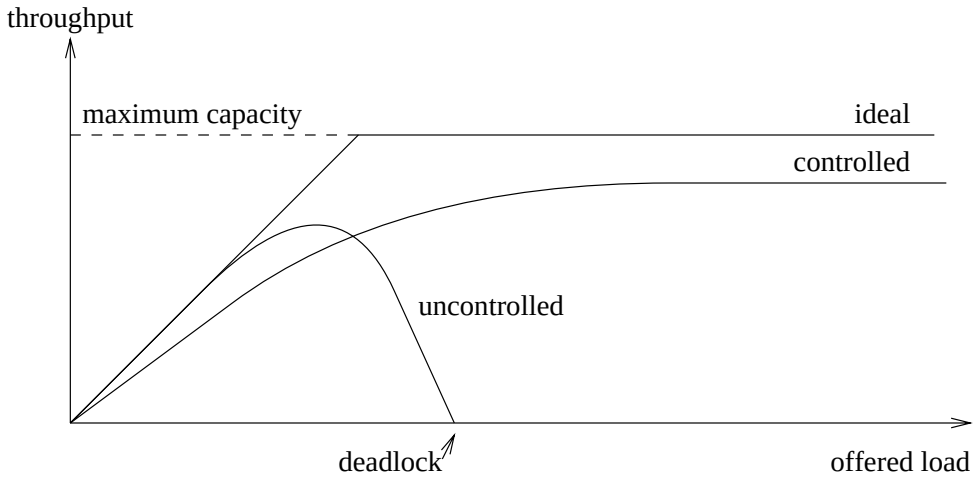


Figure 6.5. Flow control performance trade-offs

6.2.1.5. Levels of Flow Control

Since flow control levels are closely related to network protocol levels we identify the flow control structure as in Figure 6.6.

- At the physical level no flow control is assigned
- At the data link level we distinguish two types of link protocol: the node-to-node protocol and the network access protocol corresponding to node-to node flow control and network access flow control respectively.
- At the packet level protocol we have virtual circuit protocol and correspondingly the VC flow control. However, if the virtual circuit level were not implemented the network access flow control would take over.
- Above the packet protocol but within the subnet we find the entry-to-exit protocol and ETE flow control.
- Next we have the transport protocol and flow control.

6.3. Hop Level Flow Control

The objective of hop level flow control (or node-to-node) is to prevent store-and-forward buffer congestion. Hop level flow control operates locally. It monitors queues and buffer levels at each node and rejects traffic arriving at the node when some predetermined thresholds are reached.

It has been observed that a fundamental distinction between different flow control schemes here is based on the way traffic is divided into classes. We will consider 3 families of protocols: channel queue limit, buffer class and virtual circuit hop level schemes.

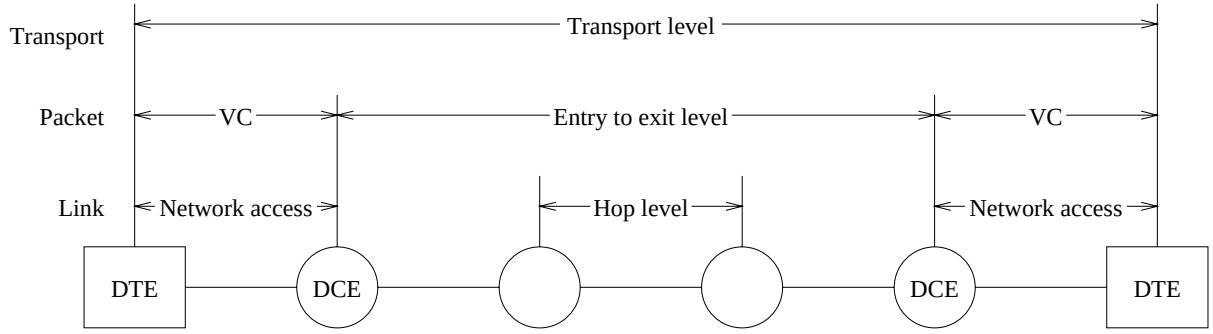


Figure 6.6. Flow control levels

6.3.1. Channel Queue Limit Flow Control

In this family the number of classes is equal to the number of channel output queues. The flow control scheme supervises and puts limits on the number of buffers each class can seize; packets beyond the limits are discarded. Within this family we have

(1) Complete Partitioning (CP)

Let B denote the buffer size, N the number of output queues and n_i the number of packets on the ith queue. The constraint for CP is

$$0 \leq n_i \leq \frac{B}{N} \quad ;\forall i$$

(2) Sharing with Maximum Queues (SMXQ)

The constraints for SMXQ are

$$0 \leq n_i \leq b_{max} \quad ;\forall i$$

$$\sum n_i \leq B$$

where $b_{\max}$ is the max queue size allowed and $b_{\max} > N$.

(3) Sharing with Minimum Allocation (SMA)

To prevent any output queue from starving, each queue is guaranteed a minimum number of buffers, $b_{\min}$ ($b_{\min} < B/N$). The constraint is then

$$\sum_i \max(0, n_i - b_{min}) \leq B - N b_{min}$$

(4) Sharing with Minimum Allocation and Maximum Queue.

This scheme combines 2 and 3.

Some form or another of CQL flow control is found in every network implementation. The ARPANET IMP has a shared buffer pool with minimum allocation and maximum limit for each

queue as shown in Figure •.

Irland found that the optimal value of the maximum queue length b_{max} is a complicated function of the mean traffic. However, he discovered that the *square root scheme* where

$$b_{max} = \frac{B}{\sqrt{N}}$$

gives, not optimal, but good performance.

It can be easily seen that the CQL flow control eliminates direct S/F deadlock.

6.3.2. Structured Buffer Pool (SBP) Flow Control

In this method, packets arriving at each node are divided into classes according to the number of hops they have covered. Consider a subnet with N nodes in which the longest route from any source to any destination is of length H hops. Each node needs $H + 1$ buffers numbered from 0 to H. The "buffer graph" is constructed (Figure 6.8) by drawing an arc from buffer i in each node to buffer $i - 1$ in each of the adjacent nodes. The legal routes from buffer i at node A are those to a buffer labelled $i + 1$ at nodes adjacent to A, and then to a buffer labelled $i + 2$ at nodes two hops away from A, etc.

An input packet from a host can only be admitted to the subnet if buffer 0 at the source node is empty. Once admitted, this packet can only move to a buffer labelled 1 in an adjacent node, and so on, until either it reaches its destination and is removed from the subnet, or it reaches a buffer labelled i, which is still not the destination, and is discarded.

This scheme can be shown to prevent "indirect store-and-forward" deadlock. The indirect store-and-forward lockup situation is illustrated in Figure 6.9. Here each node is trying to send to a neighbour, but none has any buffers available to receive incoming packets.

To see that this scheme prevents deadlock, consider the state of all buffers labelled E at some instant. Each buffer is in one of the three states

- empty
- holding a packet destined for a local host; the packet is delivered to the host.
- holding a packet for another host; this packet is looping and is dropped.

In all three cases the buffer can be made free. Consequently, packets in buffers labelled $H - 1$ can now either be delivered if it reaches its destination, or be moved forward to buffers labelled H. Thus eventually, all packets can be delivered or discarded, free of deadlocks.

Merlin and Schweitze also presented a number of improvements to this scheme to reduce the number of buffer needed and to improve the throughput efficiency:

- A packet that has already made i hops can be put in any available higher numbered buffer at the next hop not just in buffer $i + 1$.

Total buffer pool is 40 buffers

	min allocation	max allocation
Reassembly	10	20
Input queue	2	
Output queue	1	8
Total S/F buffer		20

Figure •. Buffer allocation in ARPANET

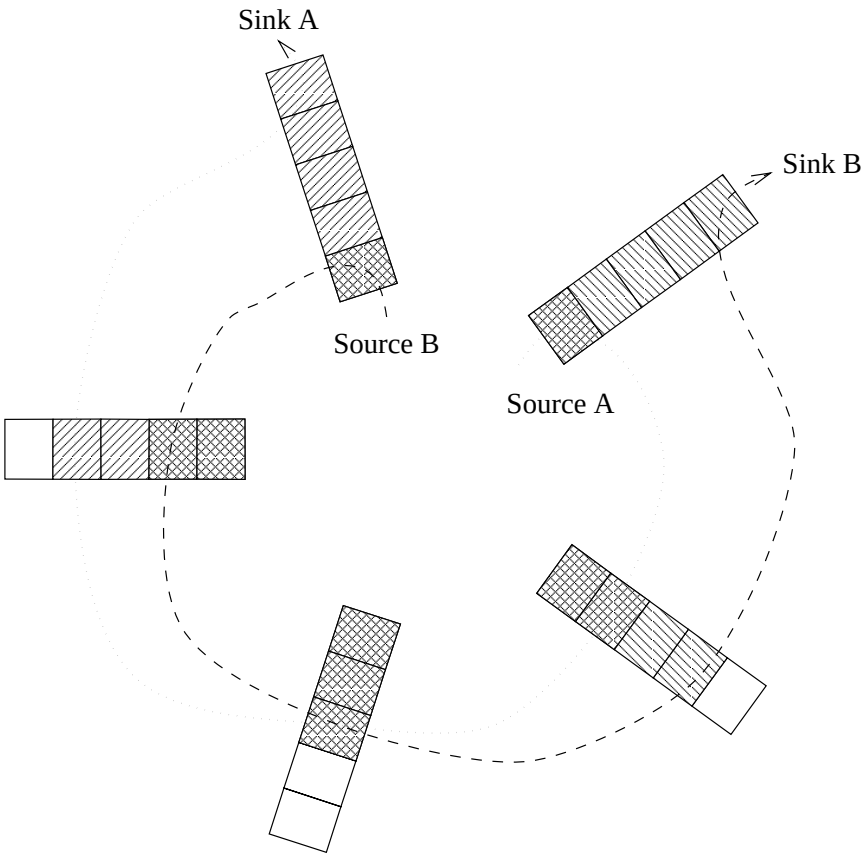


Figure 6.8.

- Limiting the number of class 0 buffers that could be seized by input packets (i.e. packets entering the network the first time from external sources).

6.3.3. Virtual Circuit Hop Level Flow Control

This family of flow control is available only in virtual-circuit networks. In VC networks, a physical network path is set up for each user session. All the packets are then forwarded along the pre-established path which is released when the session is terminated. The VC nets permit "selective flow control" to be exercised on each individual virtual circuit.

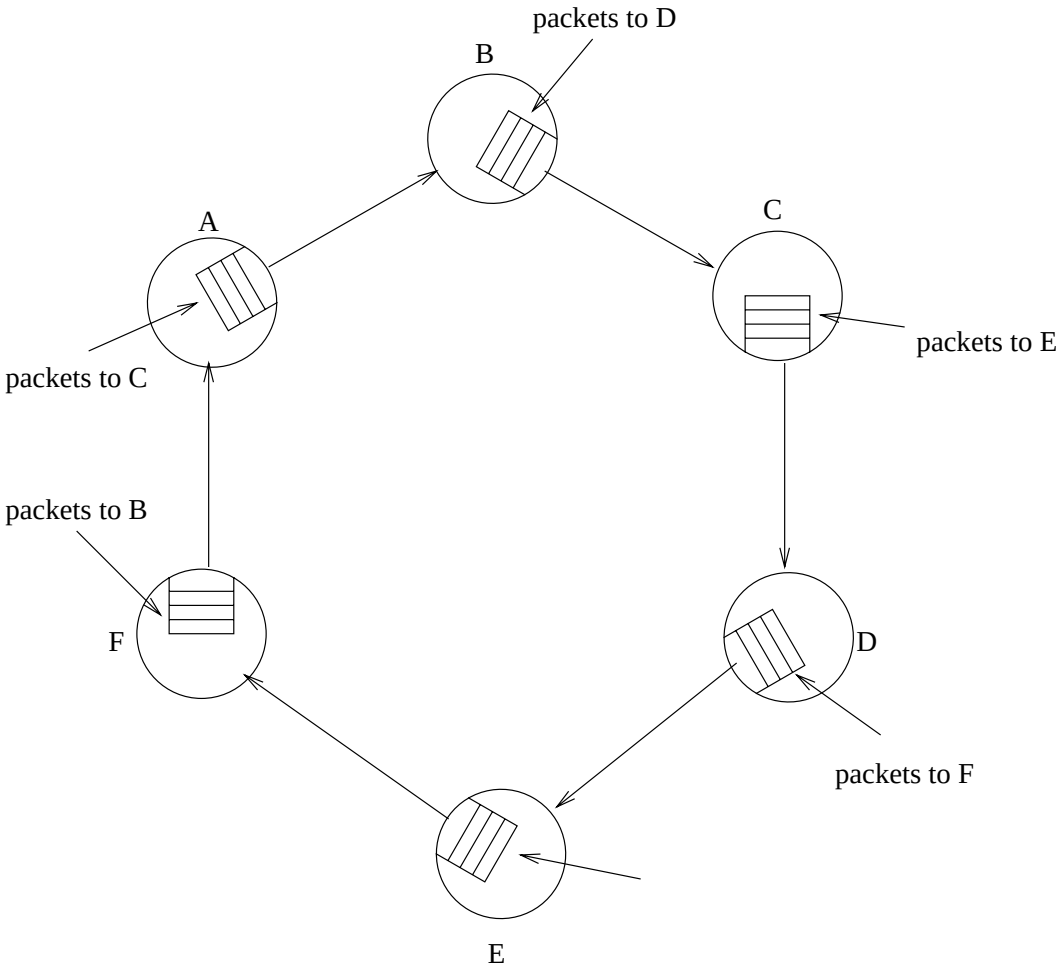


Figure 6.9. Indirect store-and-forward deadlock

The VCHL flow control procedure consists of setting up a limit L on the maximum number of packets for each VC connection that can be in transit at each intermediate node. The limit L may be fixed at set up time or may be adjusted dynamically, based on load variations. The network may refuse a connection if requirements cannot be met. One plus point about the scheme is that if a resource becomes congested, the connections (VCs) that use that resource can be traced back to their oriains and regulated at the network access level, and leaving other sources undisturbed.

In TRANSPAC network, each node records the aggregated "declared throughput" (i.e. data rate) requirements by users, and at the same time monitors actual throughput and average buffer utilization. Based on the actual to declared throughput ratio, the node may decide to "oversell" capacity, i.e. temporarily carries input rates exceeding the line capacities. VCHL flow control is exercised as follows. Packet buffers are dynamically allocated to VCs based on demand. Actions take place according to the three defined threshold levels: S_0, S_1 and S_2.

- S_0 : no further new VC requests are accepted.
- S_1 : slow down the flow on current VCs, by delaying the return of ACKs.
- S_2 : selectively disconnect existing VCs.

The threshold levels S_0, S_1 and S_2 are dynamically evaluated as a function of declared throughput, measured throughput and current buffer utilization.

6.4. Network Access Flow Control

The objective of network access flow control is to throttle external inputs to prevent overall internal buffer congestion. Congestion may be local (at the entry node), global (entire network) or selective (particular path leading to a given destination). Flow control is exercised to regulate the access of external traffic based on the condition determined at the network access points.

We will consider three NA flow control schemes: isarithmic, input buffer limit and choke packet.

6.4.1. The Isarithmic Scheme

Since the main cause of congestion is the excessive number of packets existing in the network, the Isarithmic scheme controls congestion by keeping the number of packets constant, below the congestion point.

There exist permits which circulate about within the subnet. Whenever a node wants to send a packet just given to it by its host, it must first capture a permit and destroy it. When the destination node removes the packet from the subnet, it regenerates the permit. These rules ensure that the number of packets in the subnet will never exceed the total number of permits P initially allowed.

Results of simulation studies by the National Physical Laboratories are summarized below.

- The scheme performs well in uniform traffic pattern situations, but poor in the case of nonuniform, time varying traffic patterns.
- The method of how to distribute the permits within the subnet is far from obvious.
- If permits ever get lost for any reason, the capacity of the network will be forever reduced.
- Experimental results show that the maximum number of permits that can be accumulated at each node is 3 and that $P = 3N$, where N is the total number of nodes in the subnet.

6.4.2. Input Buffer Limit Scheme

The input buffer limit (IBL) is a local network access method, whereby packets are differentiated into input and transit at the entry node. The function of IBL flow control is to throttle the input traffic when certain buffer utilization thresholds are reached in the entry node.

We will describe 3 versions of IBL flow control scheme.

(1) The IBL scheme proposed for GMDNET is a by-product of the nested buffer class structure considered previously. Input traffic is assigned to class zero and is entitled to use buffer class

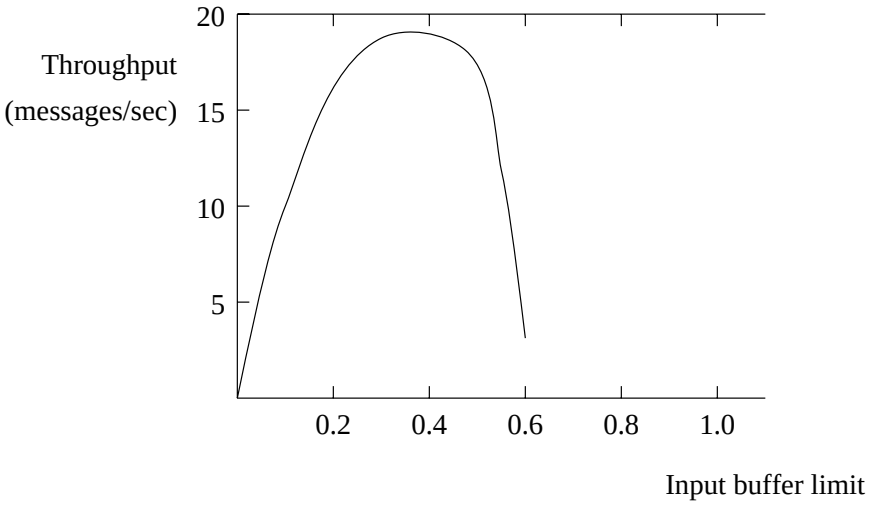


Figure 6.10. IBL Throughput vs $\frac{N_I}{N_T}$ for $N_T = 40$ (approximate)

zero only. Simulation results indicate that for a given topology and traffic pattern there is an optimal input buffer limit size wh ch maximizes throughput for heavy offered load.

(2) Another version of IBL flow control scheme was proposed and analyzed by Lam. Each node has a pool of N_T buffers, all of which may be occupied by transit traffic. However, no more than N_I ($< N_T$) buffers can be occupied by input packets. The analytical results also show that there is an optimal ratio N_I/N_T, which maximizes throughput for heavy load, as shown in Figure 6.10.

The IBL scheme is robust to external perturbations when operating at the optimal value of N_I/N_T. However, the assumption that all nodes in the subnet have the same blocking capacity is not quite realistic.

(3) Drop-and-throttle flow control (DTFC) [Kamoun]

In this version of IBL the input packet is discarded when the total number of packets in the entry node exceeds a given threshold. Transit packets, instead, can freely claim all the buffers. However, if the node is full the transit packets too will be dropped and lost. A similar scheme, called free flow scheme, is described and analyzed by Schwartz and Saad.

6.4.3. Choke Packet Scheme

In this scheme the flow control mechanism is invoked only when the system is congested. The mechanism described below was proposed for the Cyclades network.

Each node monitors the utilization percentage of each of its output lines. Associated with each line is a real variable, u_k, whose value, between 0.0 and 1.0, reflects the recent utilization of that line at time k; and f_k (either 0 or 1) the instantaneous line utilization. u_k is updated according to

$$u_k = au_{k-1} + (1-a)f_k$$

where the constant a determines how fast the node forgets recent history.

Whenever u_k moves above the threshold, the output line enters a warning state. Each newly arriving packet is checked to see if its output line is in warning state. If so and

- If the packet is an input packet it is dropped.
- If the packet is a transit packet

  the node sends a choke packet back to the source node giving it the destination found in the transit packet. The transit packet is tagged so that it will not generate any more choke packets later, and is forwarded on the path.

  When the source node gets the choke packet, it reduces the traffic sent to the specified destination by X%. The source host then ignores other choke packets for a fixed time interval T seconds.

  After T seconds the source host listens for more choke packets for another interval. If one arrives, the line is still congested, so the source host reduces the flow still further and repeat the same process. If no more choke packets arrive during the listening period, the source host may increase the flow again.

Simulation results based on the Cigule network topology are given in Figure 6.11.

The Choke Packet Scheme is different from the IBL scheme in that it uses a path congestion measure to exercise selective flow control on input traffic directed to different destinations.

6.5. Entry-to-Exit Flow Control

The objective of the entry-to-exit (ETE) flow control is to prevent buffer congestion at the exit (or destination) node due to the fact that the remote sources are sending traffic at a higher rate than can be accepted by the destination hosts.

If message sequencing and/or reassembly is required, the entry to exit scheme is necessary to avoid deadlocks at the exit node.

Examples of deadlocks at the exit node are given in Figure 6.12.

Virtually all ETE controls are based on a window scheme that allows only up to W sequential messages to be outstanding in the network before an end-to-end ACK is received.

We will describe a few ETE schemes implemented by various networks.

6.5.1. Arpanet RFNM and Reassembly Scheme

ETE flow control in ARPANET applies to each pair of hosts. Sliding window scheme is used: a limit of eight messages between each pair of hosts. All messages travelling from the same host to the same destination are numbered sequentially. At the destination, messages whose sequence numbers are out of the window range are discarded. Messages arriving out of order are also discarded. When a message is received correctly a Ready For Next Message (RFNM) is acknowledged back to the sender so that it can advance its sending window. If there has been no response for 30 seconds the source node sends a query to the destination node. Retransmission

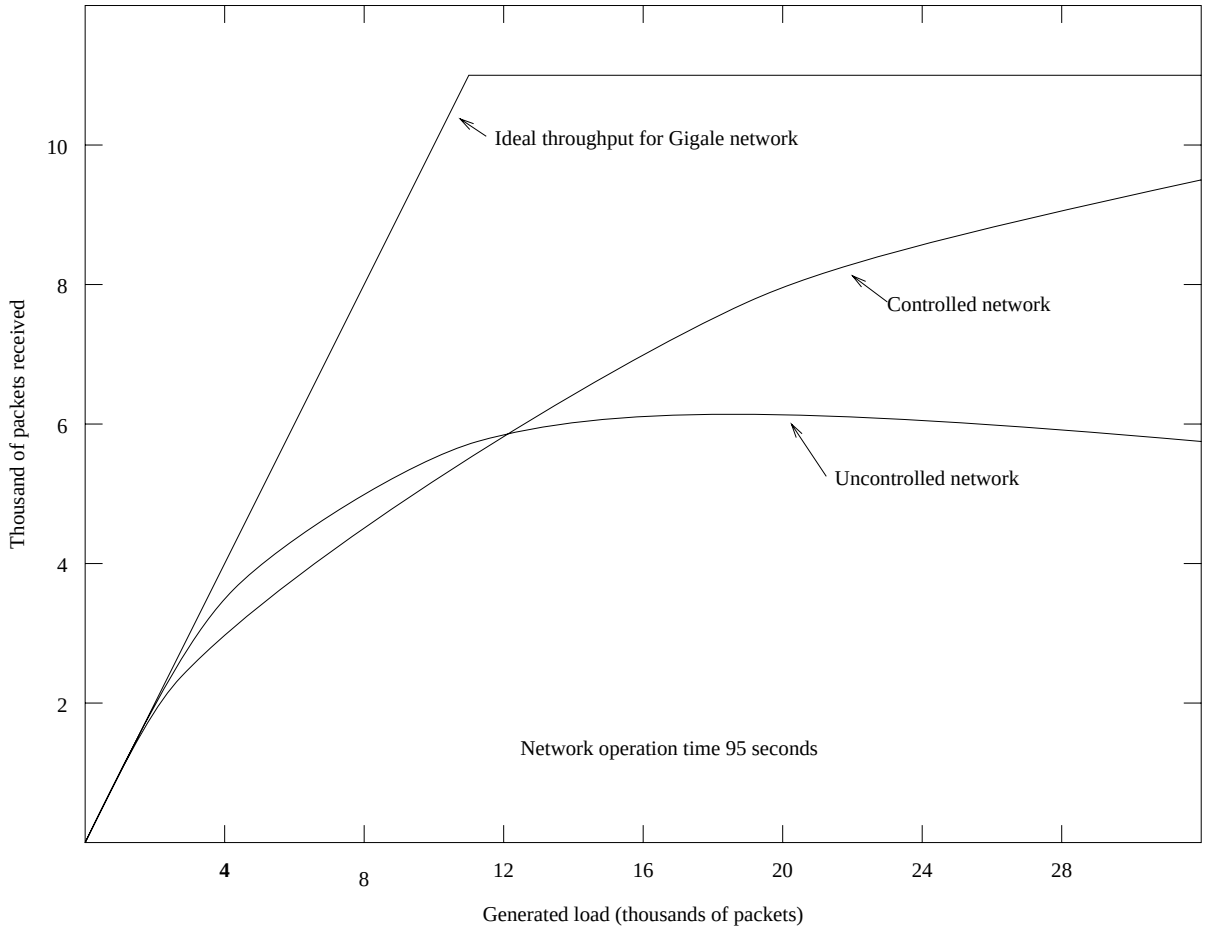


Figure 6.11. Throughput performance in Cigale with and without flow control (approximate)

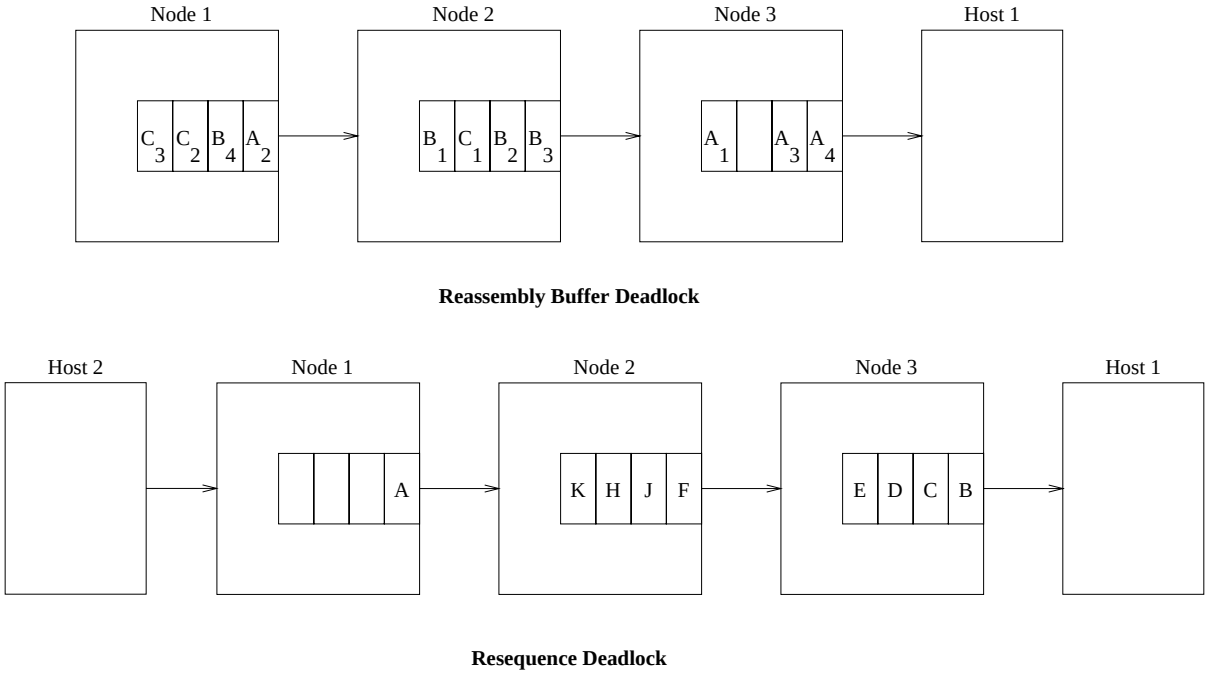


Figure 6.12.

of the message might be required depending on the answer.

However, deadlock still occurs if there are not enough buffers to reassemble all the packets of a multipacket message at the destination. The ARPANET solved the deadlock problems by using following mechanisms:

(a) Reservation for Multipacket Messages.

Before a source node sends a message to a destination node, it is required to send a REQ ALL (request for allocation) message to the destination. Only after receiving an ALL (allocate) packet it is allowed to send the message. If the destination node does not have enough buffers, it just queued the request and sent ALL packet later and in the meantime the source node just has to wait.

To be more efficient and to reduce the delay, the next lot of buffers is automatically reserved for the same stream of messages by the destination node. This is done by piggybacking an ALL packet on an RFNM back to the source. If the source host failed to send another multipacket message within 250 msec, the source node would give back the allocation.

(b) Reservation for Single-Packet Message

A single packet message serves as its own allocation request. If the destination node has room, it accepts the message and sends back an RFNM immediately. If it does not have any room, it discards the message and queues the request. It notifies the source node later when space becomes available.

6.5.2. SNA Virtual Route Pacing Scheme

Flow control is done per virtual route. This scheme is called "VR pacing". It is actually a combination of ETE and hop to hop level flow control. Upon receipt of transmit authorization from the destination, the source is permitted to send a window of N packets, the first of which will normally contain a request to send N more packets. When the destination (exit node) has sufficient buffer space available, it returns an authorization for the next group of M packets.

Congestion control is achieved by dynamically adjusting the pacing parameter, N. For each virtual route N can vary from h to $3h$, where h is the number of hops in the route. Each packet contains 2 bits that are set to 0 by the source node. An intermediate node can set one bit or the other to 1 depending on how congested the route becomes. If the route is not congested the bits are left alone. When each packet arrives at its destination, the state of the bits tells how much congestion there is along the path. Based on the amount of congestion, the parameter N for the corresponding virtual route can be increased or decreased.

Chapter 7. Introduction to Queueing Theory

"They also serve who only stand and wait".
MILTON.

"Ah, 'All things come to those who wait'.
They come, but often come too late".
From Lady Mary M. Currie: Tout Vient a Qui Sait Attendre, 1890.

7.1. Introduction

Think for a moment how much time is spent in one's daily activities waiting in some form of a queue: stopped at a traffic light; delayed at a supermarket checkout stand; standing in line for a ticket at a box-office; holding the telephone as it rings, and so on. These are only a few examples of queueing systems.

The thing common to all the systems that we shall consider is a flow of *customers* requiring *service*, there being some restriction on the service that can be provided. For example, the customer may be aircraft requiring to take-off, the restriction on 'service' being that only one aircraft can be on the runway at a time. Or, the customers may be patients arriving at an out-patient's clinic to see a doctor, the restriction on service is again that only one customer can be served at a time. The two examples are both cases of *single-server queue.*

An example of a *multi-server queue* is a queue for having goods checked at a supermarket; here the restriction is that not more than, say, *m* customers can be served at a time.

In this section we study the phenomena of standing, waiting and serving – *queueing theory.*

7.2. Aims and Characterisations

In the examples given so far, the restriction on service is that not more than a limited number of customers can be served at a time, and congestion arises because the unserved customers must queue up and await their turn for service. For example, if the customers and messages waiting to be served at a concentrator in a computer network, the messages arrive irregularly due to the bursty nature of the traffic. Then from time to time more than one customer will be at the service point at the same time, all but one of them (in a single-server queue) must queue up in the buffer awaiting their turn for service, and congestion has occurred. This simple point illustrates an important general principle, namely that the congestion occurring in any system depends in an essential way on the irregularities in the system and not just on the average properties.

Our aim in investigating a system with congestion is usually to improve the system by changing it in some way. For example, the rate of arrival of messages may be so high that large queues

develop, resulting in a long delay-time per message passing through a computer network; or the rate of arrival may ke so low that the concentrators are unused for a large proportion of time. In either case a change in the system may be economically profitable. In any case, it is often very helpful to be able to predict what amount of congestion is likely to occur in the modified system.

In order to investigate congestion we must specify the queueing system sufficiently. Generally a queueing system can be characterized by the following components:

1. The arrival pattern. This means both the average rate of arrival of customers and the statistical pattern of the arrivals. Generally, the arrival process is described in terms of the probability distribution of the interval between consecutive arrivals. The distribution is denoted by $A(t)$, where

$$A(t) = P[\text{time between arrivals} < t]$$

2. The service discipline. This means

   (a) The service time, which refers to the length of time that a customer spends in the service facility. The probability distribution of the service time is denoted by $B(x)$, where

$$B(x) = P[\text{service time} < x]$$

   (b) The number of servers, m.

3. The queueing discipline. This describes the order in which customers are taken from the queue. The simplest queueing discipline consists in serving customers in order of arrival, i.e. first-come-first-serve (FCFS), but there are many other possibilities.

4. The amount of buffer space in the queues. Not all queueing systems have an infinite amount of buffer space. When too many customers are queued up for a finite number of slots, some customers get lost or rejected.

The notation $A/B/m$ is widely used for the specification of queueing systems, where A is the interarrival-time probability distribution, B the service-time probability distribution, and m the number of servers. The probability distributions A and B are often chosen from the set

M exponential probability density (M: Markov)

D all customers have the same value (D: Deterministic)

G arbitrary distribution (G: General)

Thus, for example, the system $M/M/1$ is a single-server queueing system with exponentially distributed interarrival times, exponentially distributed service times.

We sometimes need to specify the system's buffer storage (which we denote by K) or to specify the customer population (which we denote by M) and in these cases the notation $A/B/m/K/M$ is used. If either of these last 2 parameters is absent, then we assume it takes on the value of infinity. For example, the system $M/D/3/10$ is a 3-server system with exponentially distributed interarrival times, with constant service times and with a system storage capacity of size 10.

We have indicated how one must specify a queueing system; we now identify some of the properties of a queueing system that may be of practical interest. Basically we are interested in

(1) the mean and distribution of the length of time for which a customer has to queue for service, i.e. the *waiting time*;

(2) the mean and distribution of the number of customers in the system at any instant;

(3) the mean and distribution of the length of the server's busy periods.

These three properties of the queueing system are related in a general way, in that all three mean values tend to increase as a system becomes more congested. In order to study the system analytically we will describe the structure of basic queueing systems in the next section.

7.3. The structure for basic queueing systems

We consider a very general queueing system $G/G/m$. We focus attention on the flow of customers as they arrive, pass through and eventually leave the system. We portray our system as in Figure 7.1, where

C_n the nth customer to enter the system

$N(t)$ the number of customers in the system at time t.

We define a number of quantities for the general queueing system as follows (some of these quantities are illustrated in Figure 7.2):

τ_n arrival time for C_n

t_n interarrival time between C_n and $C_n - 1$ (i.e. $\tau_n - \tau_{n-1}$.
$\bar{t} \equiv \dfrac{1}{\lambda}$ – the average interarrival time

$A(t)$ the interarrival times probability distribution
$= P[t_n \leq t]$ independent of n

x_n service time for C_n

$\bar{x}$ average service time per customer $= \dfrac{1}{\mu}$

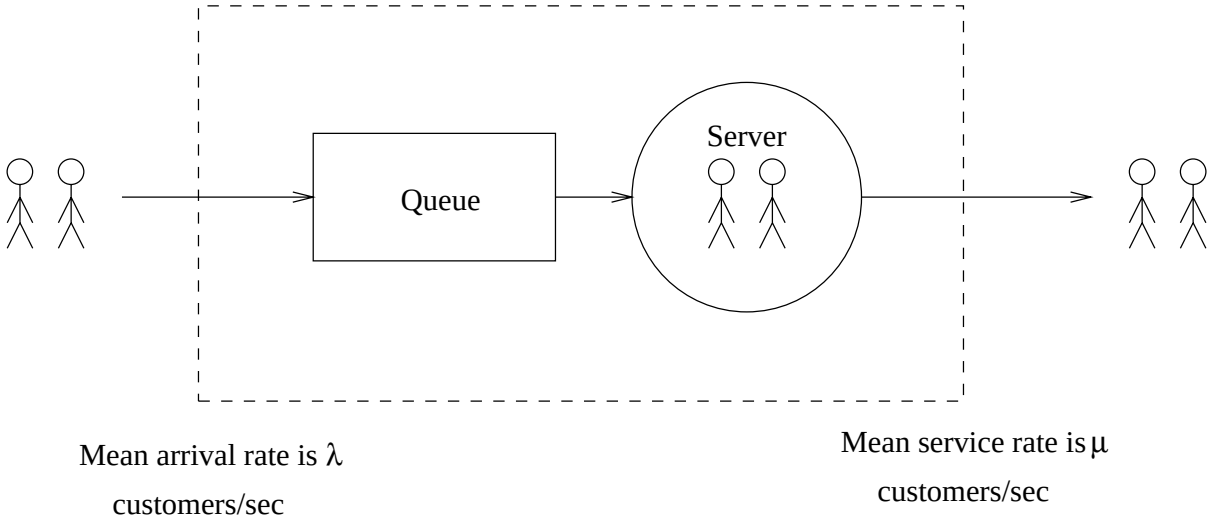


Figure 7.1.

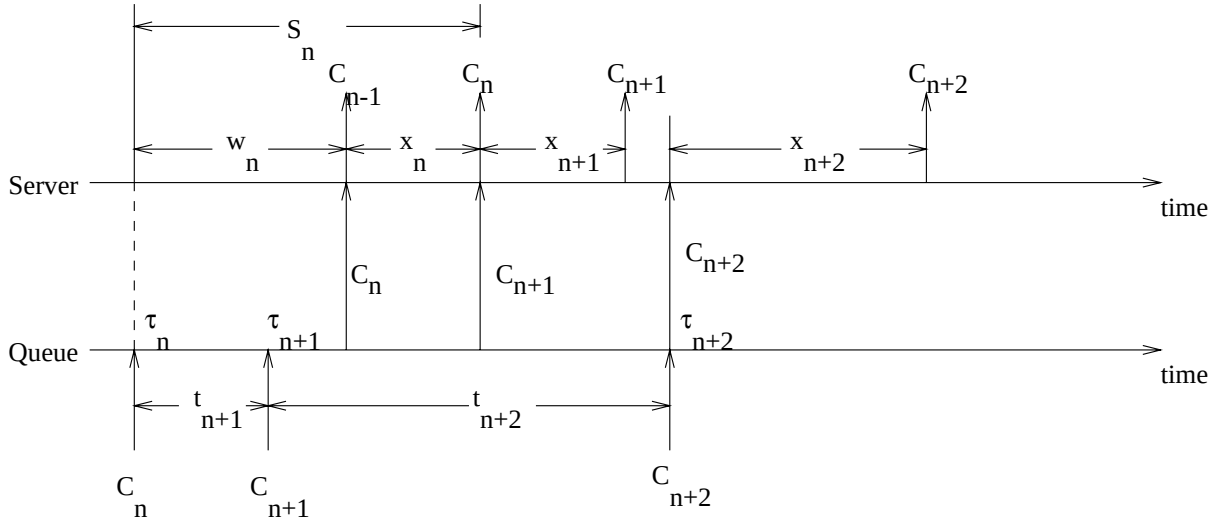


Figure 7.2.

$B(x)$ the service time probability distribution
$= P[x_n \leq x]$ independent of n

W_n waiting time (in queue) for C_n

W average waiting time per customer

S_n total delay time or system time for C_n
$= W_n + X_n$

T average system time
$= W + \bar{x}$

ρ the traffic intensity or the utilization factor

$$= \frac{\text{average arrival rate of customers}}{\text{average service rate of the servers}}$$

$= \frac{\lambda}{\mu}$ or $\lambda\bar{x}$ for single server system

$= \frac{\lambda}{m\mu}$ or $\frac{\lambda\bar{x}}{m}$ for m-server system

Little's theorem: the average number of customers in a queueing system is equal to the average arrival rate of customers to that system, times the average time spent in that system, i.e.

$$\overline{N} = \lambda T$$

7.4. The arrival pattern

In this section we will describe some common types of arrival pattern that are most used in the queueing theory.

(a) Regular arrival

The simplest arrival pattern physically is the regular one in which customers arrive simply at equally spaced instants, T units of time apart. The rate of arrival of customers is $\lambda = \frac{1}{T}$ per unit time. A practical example where the arrivals is nearly regular is in an appointment system.

(b) Completely random arrivals or Poisson arrivals

The simplest arrival pattern mathematically, and the most commonly used in all applications of queueing theory is the completely random arrival process. To define this process formally, let λ be a constant representing the average rate of arrival of customers and consider a small time interval Δt, with $\Delta t \to 0$. The assumptions for this process are as follows:

- The probability of one arrival in an interval of Δt sec, say $(t, t + \Delta t)$ is $\lambda\Delta t$, independent of arrivals in any time interval not overlapping $(t, t + \Delta t)$.
- The probability of no arrivals in Δt sec is $l - \lambda\Delta t$. Under such conditions, it can be shown that the probability of exactly k customers arriving during an interval of length t is given by the Poisson law:

$$P_k(t) = \frac{(\lambda t)^k e^{-\lambda t}}{k!} \quad k \geq 0, t \geq 0$$

The mean of the distribution

Defining k as the number of arrivals in this interval of length t we have

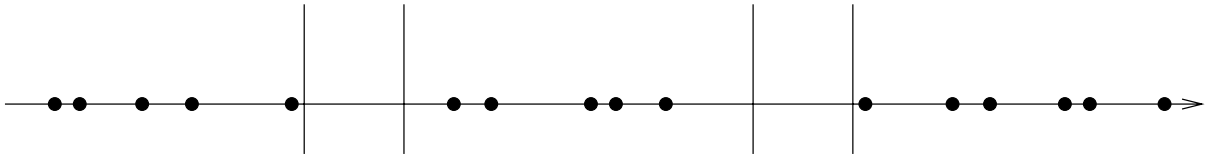


Figure 7.3. Fundamental property of the Poisson arrivals. Two intervals A and B of equal length and therefore with equal probability for an arrival.

$$E[k] = \sum_{k=0}^{\infty} kP_k(t)$$

$$= e^{-\lambda t} \sum_{k=0}^{\infty} k \frac{(\lambda t)^k}{k!}$$

$$= E^{-\lambda t} \sum_{k=1}^{\infty} \frac{(\lambda t^k}{(k-1)!}$$

$$= e^{-\lambda t} \lambda t \sum_{k=0}^{\infty} \frac{(\lambda t)^k}{k!}$$

$$E[k] = \lambda t \quad \text{as } e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$$

The variance of the distribution

$$\sigma_k^2 = E[(k - E[k])^2]$$

$$= E[k^2 - (E[k])^2 - 2kE[k]]$$

$$= E[k^2] - (E[k])^2$$

σ_k^2 can also be rewritten as follows:

$$\sigma_k^2 = E[k(k-1)] + E[k] - (E[k])^2$$

but

$$E[k(k-1)] = \sum_{k=0}^{\infty} k(k-1)P_k(t)$$

$$= e^{-\lambda t} \sum_{k=0}^{\infty} k(k-1) \frac{(\lambda t)^k}{k!}$$

$$= e^{0-\lambda t} (\lambda t)^2 \sum_{k=2}^{\infty} \frac{(\lambda t)^{k-2}}{(k-2)!}$$

$$= e^{-\lambda t} (\lambda t)^2 \sum_{k=0}^{\infty} \frac{(\lambda t)^k}{k!}$$

$$= (\lambda t)^2$$

Thus

$$\sigma_k^2 = (\lambda t)^2 + \lambda t - (\lambda t)^2$$

$$\sigma_k^2 = \lambda t = E[k]$$

Now we will show that the time t between arrivals is a continuously-distributed exponential random variable.

By definition

$$A(t) = P[t_n \leq t]$$

$$= 1 - P[t_n > t]$$

But $P[t_n > t]$ is just the probability that no arrivals occur in $(0, t)$, that is, $P_0(t)$. Therefore, we have

$$A(t) = 1 - P_0(t)$$

i.e. the Probability Distribution Function [PDF] is given by

$$A(t) = 1 - e^{-\lambda t}; \quad t \geq 0$$

and the probability density function is given by

$$f(t) = \frac{dA(t)}{dt}$$

$$f(t) = \lambda e^{-\lambda t}, \quad t \geq 0$$

Thus the Poisson arrivals generate an exponential interarrival distribution whose pdf and PDF are shown in figure 7.4.

The mean

$$E[t] = \int_0^\infty t f(t) dt$$

$$= \int_0^\infty t \lambda e^{-\lambda t} dt$$

$$= \frac{1}{\lambda}$$

The second moment

$$E[t^2] = \int_0^\infty t^2 f(t) dt$$

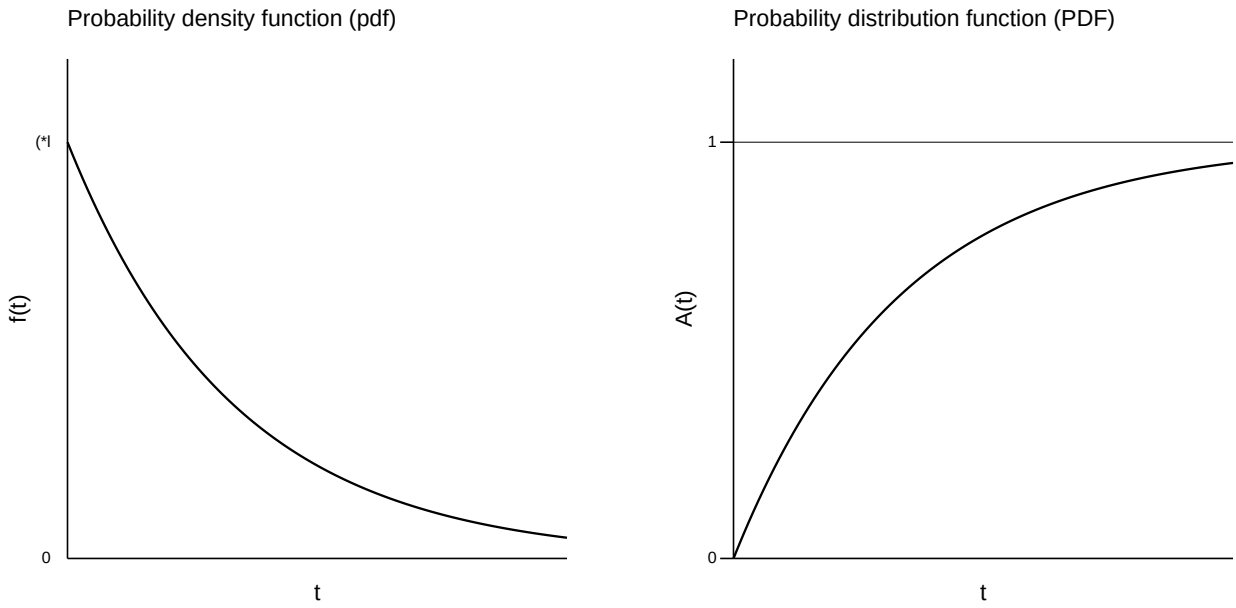


Figure 7.4.

$$= \int_0^\infty t^2 \lambda e^{-\lambda t} dt$$

$$= \frac{2}{\lambda^2}$$

The variance

$$\sigma_t^2 = E[t^2] - (E[t])^2$$

$$= \frac{2}{\lambda^2} - \frac{1}{\lambda^2} = \frac{1}{\lambda^2}$$

The coefficient of variation

$$C_a = \frac{\sigma_t}{E[t]} = 1$$

i.e. 100%.

7.5. The service time distribution

(a) **Constant service time**. The service time may be assumed to be constant. This is always an idealization but often gives adequate answers.

(b) **Exponential service time**. Again we can assume that the service time distribution is exponential with probability density function $B(x)$, where

$$B(x) = \mu e^{-\mu x}, \quad x \geq 0$$

The average number of customers served per second is V and the average service time per customer is

$$E[x] = \frac{1}{\mu}$$

The service thus is treated as if it were a complete random operation with the property that the probability that service is completed in a small element of time is constant ($= \mu\Delta x$), independent of how long service has been in progress.

Chapter 8. Simple Queues with Random Arrivals

In this chapter we consider a number of simple queueing systems. We will also present some methods that can be used in dealing with these queueing problems. We shall focus only on the equilibrium behaviour of these systems.

8.1. Equilibrium solutions

Simple example

Consider a system which at any instant is in one of two possible states, A and B as shown in Figure 8.1.

We assume

- The probabilities of finding the system in state A and state B at time t are $p_A(t)$ and $p_B(t)$ respectively.
- The probability that the system moves from state A at time t to state B at time $(t + \Delta t)$ is $\lambda\Delta t$; and that from B to A is $\mu\Delta t$.

With these assumptions the probability that the system is in state A at time $(t + \Delta t)$ is the sums of the probabilities that

(a) the system is in state A at time t and no transition occurs in the interval $(t, t + \Delta t)$.

(b) the system is in state B and a transition from B to A occurs in $(t, t + \Delta t)$.

That is

$$p_A(t + \Delta t) = (1 - \lambda\Delta t)p_A(t) + \mu\Delta t p_B(t) \tag{8.1.1}$$

Similarly,

$$p_B(t + \Delta t) = \lambda\Delta t p_A(t) + (1 - \mu\Delta t)p_B(t) \tag{8.1.2}$$

Letting $\Delta t \to 0$, we have

$$\frac{d}{dt}p_A(t) = -\lambda p_A(t) + \mu p_B(t) \tag{8.1.3}$$

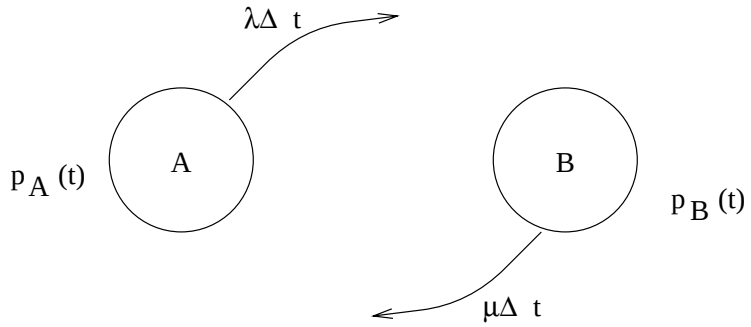


Figure 8.1.

$$\frac{d}{dt}p_B(t) = \lambda p_A(t) - \mu p_B(t)$$

Now in equilibrium the probability of finding the system in a given state does not change with time, i.e.

$$\frac{d}{dt}p_A(t) = \frac{d}{dt}p_B(t) = 0 \tag{8.1.4}$$

and this does not mean that transitions do not occur within the system, it simply means that at each state of the system the two transitions, transition away from the state and transition to the state, must occur at the same rate. From equations 8.1.3 and 8.1.4 we can see that (as shown in Figure 8.2)

$$\lambda p_A = \mu p_B \tag{8.1.5}$$

From equation 8.1.5 and the condition that

$$p_A + p_B = 1 \tag{8.1.6}$$

we can solve for P_A and P_B:

$$p_A = \frac{\mu}{\lambda + \mu} \text{ and } p_B = \frac{\lambda}{\lambda + \mu} \tag{8.1.7}$$

It should be noted that equation 8.1.5 can be written down simply by observing the state diagram of figure 8.2, saving all the trouble in writing down (8.1.2) and (8.1.3) first.

General Situation

Consider a general system with a set of states $A_0, A_1, A_2 \ldots$ where $p_k(t)$ denotes the probability of finding the system in state A_k at time t. We suppose that the probability that the system jumps from the state A_k at time t to the state A_{k+1} at time $(t + \Delta t)$ is $\lambda_k \Delta t$, independent of the history of the system before time t. Likewise the system may jump from A_k at time t to A_{k-1} at time $(t + \Delta t)$

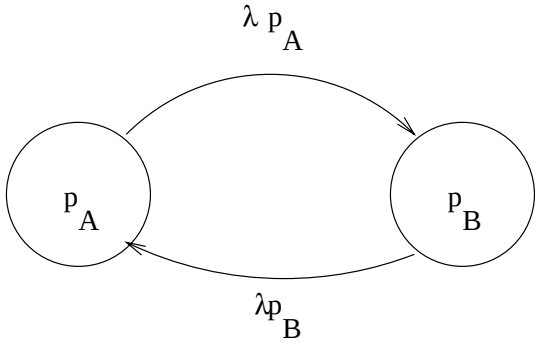


Figure 8.2.

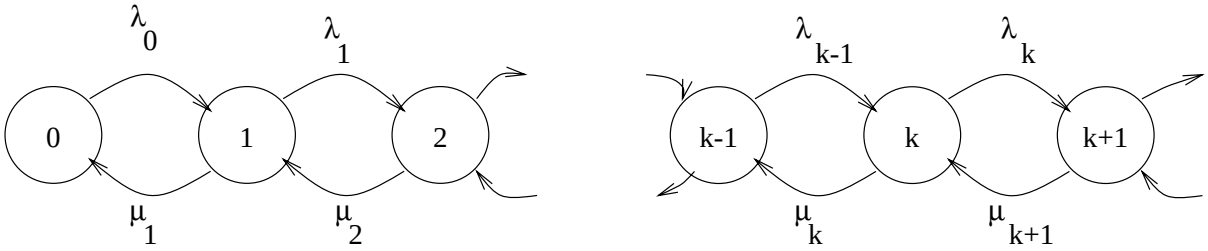


Figure 8.3.

with probability $p_k \Delta t$; and of course we must have $\mu_0 = 0$ because there is no state A_{-1}.
For any state A_k the probability $p_k(t)$ satisfies the following differential equation (analogous to (8.1.3)).

$$\frac{dp_k(t)}{dt} = -(\lambda_k + \mu_k)p_k(t) + \lambda_{k-1}p_{k-1}(t) + \mu_{k+1}p_{k+1}(t) \tag{8.1.8}$$

When the system is in equilibrium we can equate $\frac{dp_k(t)}{dt}$ to zero in equation 8.1.8 for all k and solve for $p_k(t)$ $(k = 0, 1, 2\ldots)$ subject to the condition that

$$p_0 + p_1 + p_2 + \ldots = 1 \tag{8.1.9}$$

Alternatively, with the help of the state diagram for the system (figure 8.3) we can directly write down the set of equilibrium equations.

$$\lambda_0 p_0 = \mu_1 p_1 \tag{8.1.10}$$

$$(\lambda_1 + \mu_1)p_1 = \lambda_0 p_0 + \mu_2 p_2$$

or in general

$$(\lambda_k + \mu_k)p_k = \lambda_{k-1}p_{k-1} + \mu_{k+1}p_{k+1} \quad (k \geq 1)$$

and again solve (8.1.10) for p_k subject to (8.1.4).

8.2. M/M/1 queue: single-server queue with random arrivals and exponential service times

In this system we have

$$\lambda_k = \lambda \quad ; k = 0, 1, 2\ldots$$

$\mu_k = \mu \quad ; k = 1, 2, 3\ldots$

We say that the state A_k is occupied if there are k customers in the queue including the one being served. Let us apply the ideas of section 8.1 to our queueing situation here. We first construct the state diagram for the system as in Figure 8.4. Then we write down the equilibrium equations for the systems as follows

$$\lambda p_0 = \mu p_1 \tag{8.2.1}$$

$(\lambda + \mu)p_1 = \lambda p_0 + \mu p_2$

$$\vdots$$

$$(\lambda + \mu)p_k = \lambda p_{k-1} + \mu p_{k+1}$$

with the normalizing condition

$$p_0 + p_1 + \ldots = 1 \tag{8.2.2}$$

The recursive solution for (8.2.1) is

$$\lambda p_k = \mu p_{k+1} \tag{8.2.3}$$

or in terms of p_0 and $\rho = \dfrac{\lambda}{\mu}$

$$p_k = \rho^k p_0 \tag{8.2.4}$$

To find p_0 we use equations (8.2.2) and (8.2.4)

$$\sum_{k=0}^{\infty} \rho^k p_0 = 1$$

and

$$\sum_{k=0}^{\infty} \rho^k = \frac{1}{1-\rho} \tag{8.2.5}$$

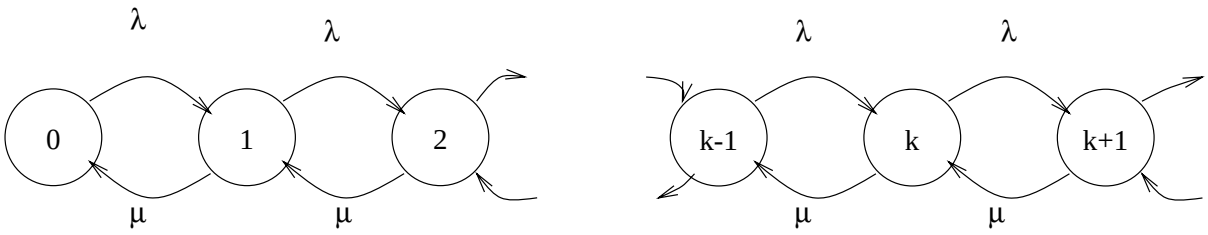


Figure 8.4.

Hence

$$p_0 = (1-\rho) \tag{8.2.6}$$

and in general

$$p_k = (1-\rho)\rho^k \tag{8.2.7}$$

The mean number of customers in the system, N, can be found directly from (8.2.6)

$$\overline{N} = \sum_{k=0}^{\infty} k p_k = (1-\rho)\sum_{k=0}^{\infty} k\rho^k$$

$$\overline{N} = \frac{\rho}{1-\rho} \tag{8.2.8}$$

Exercise: Show that $\overline{N} = \dfrac{\rho}{1-\rho}$

The variance of the number of customers in the system is found to be

$$\sigma_N^2 = \frac{\rho}{(1-\rho)^2} \tag{8.2.9}$$

Using Little's result we can now find the total average delay time, including the service time:

$$T = \frac{\overline{N}}{\lambda} = \frac{\rho/\lambda}{1-\rho} = \frac{1/\mu}{1-\rho} = \frac{1}{\mu-\lambda} \tag{8.2.10}$$

The probability of finding at least k customers in the system is

$$P[N \geq k \text{ in the system}] = \sum_{i=k}^{\infty} p_i$$

$$= (1-\rho)\sum_{i=k}^{\infty}\rho^i$$

$$= \rho^k \tag{8.2.11}$$

Figure 8.5 shows the dependence of the queue length and of the total delay time on the traffic intensity ρ.

As ρ approaches 1, both the queue length and the total delay grow very quickly.

8.3. Arrival rates and service times dependent on queue size

In this section we cover more general systems in which the rate at which customers arrive may depend on the size of the queue. We suppose that transitions $A_k \to A_{k+1}$ are associated with a probability $p_k \Delta t$ and that transitions $A_k \to A_{k-1}$ (for $k > 0$) are associated with a probability $\mu_k \Delta t$. We still assume that the probabilities referring to transition in $(t, t + \Delta t)$ do not depend on what happened before t. For this situation the set of equilibrium equations (8.1.9) applies and the solutions are found to be

$$p_k = \frac{\lambda_0\lambda_1\lambda_2\ldots\lambda_{k-1}}{\mu_1\mu_2\mu_3\ldots\mu_k}p_0 \tag{8.3.1}$$

or

$$p_k = p_0\prod_{i=0}^{k-1}\frac{\lambda_i}{\mu_{i+1}} \quad k = 0, 1, 2\ldots \tag{8.3.2}$$

By using condition (8.2.1) we obtain

$$p_0 = \frac{1}{1+\sum_{k=1}^{\infty}\prod_{i=0}^{k-1}\frac{\lambda_i}{\mu_{i+1}}} = S^{-1} \tag{8.3.3}$$

We will apply these formulas to a number of queueing systems.

8.3.1. Queue with discouraged arrivals

Consider a case where the sight of a long queue discourages customers from joining it, say (see Figure 8.6)

$$\lambda_k = \frac{\alpha}{k+1} \quad k = 0, 1, 2\ldots$$

$$\mu_k = \mu \quad k = 1, 2, \ldots$$

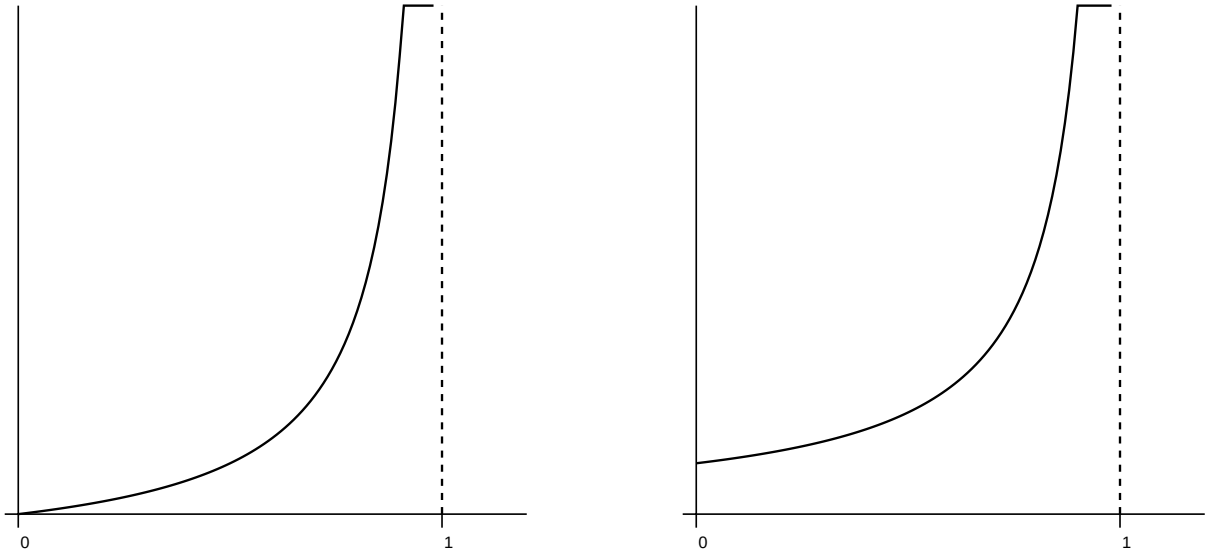


Figure 8.5.

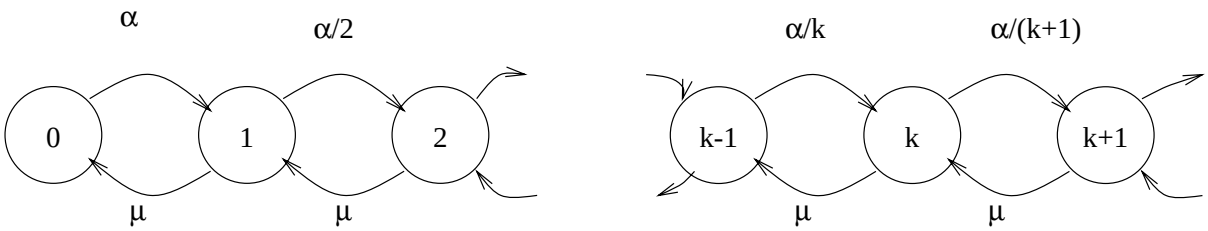


Figure 8.6. State-transition-rate diagram for discouraged arrivals

then

$$S = 1 + \frac{\alpha}{\mu} + \frac{1}{2!}\left(\frac{\alpha}{\mu}\right)^2 + \frac{1}{3!}\left(\frac{\alpha}{\mu}\right)^3 + \ldots = e^{\alpha/\mu}$$

and we find that

$$p_0 = e^{-\alpha/\mu} \tag{8.3.1.1}$$

$$p_k = \frac{(\alpha/\mu)^k}{k!} e^{-\alpha/\mu} \tag{8.3.1.2}$$

We thus have a Poisson distribution of queue size with mean $N = \frac{\alpha}{\mu}$. The probability that the server is free is $e^{-\alpha/\mu}$. The traffic intensity ρ can also be interpreted as the fraction of time the server is busy, i.e.

$$\rho = 1 - p_0 = \lambda \bar{x} \tag{8.3.1.3}$$

We have therefore

$$\rho = 1 - e^{-\alpha/\rho} \tag{8.3.1.4}$$

$$= \frac{\lambda}{\mu}$$

$$\lambda = \mu\sigma = \mu(1 - e^{-\alpha/\mu})$$

Using Little's result we obtain

$$T = \frac{\overline{N}}{\lambda} = \frac{\alpha}{\mu^2(1 - e^{-\alpha/\mu})} \tag{8.3.1.5}$$

8.3.2. M/M/∞: Queue with infinite number of servers

We consider the case that whenever a customer arrives there is a server made available to deal with him without delay (Figure 8.7).

$$\lambda_k = \lambda \quad k = 0, 1, 2\ldots$$

$$\mu_k = k\mu \quad k = 1, 2, 3\ldots$$

Here we find

$$S = 1 + \frac{\lambda}{\mu} + \frac{1}{2!}\left(\frac{\lambda}{\mu}\right)^2 + \frac{1}{3!}\left(\frac{\lambda}{\mu}\right)^3 + \ldots = e^{\lambda/\mu}$$

$$p_0 = e^{-\lambda/\mu}$$

$$p_k = \frac{(\lambda/\mu)^k}{k!} e^{-\lambda/\mu} \tag{8.3.2.1}$$

Again this is a Poisson distribution with mean queue size $\overline{N} = \frac{\lambda}{\mu}$ and average delay

$$T = \frac{\overline{N}}{\lambda} = \frac{1}{\mu} \tag{8.3.2.2}$$

8.3.3. M/M/m: Queue with m servers

In this case (see figure 8.8)

$$\lambda_k = \lambda \quad k = 0, 1, 2\ldots$$

$$\mu_k = k\mu \quad 0 \leq k \leq m$$

$$\mu_k = m\mu \quad k \geq m$$

Here we find

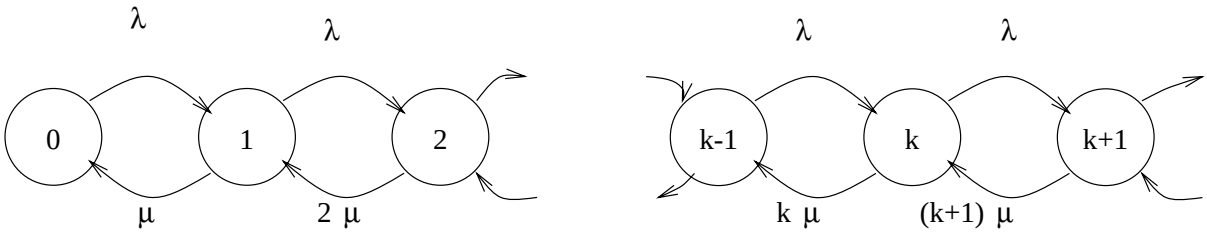


Figure 8.7.

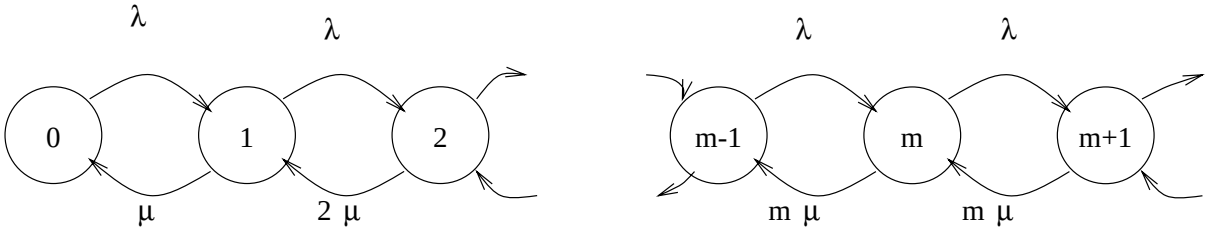


Figure 8.8.

For $k < m$

$$p_k = S^{-1}\prod_{i-0}^{k=i}\frac{\lambda}{\mu(i+1)}$$

$$= S^{-1}\left(\frac{\lambda}{\mu}\right)^k\frac{1}{k!}$$

$$p_k = S^{-1}\frac{(m\rho)^k}{k!}; \quad \rho = \frac{\lambda}{m\mu} < 1 \tag{8.3.3.1}$$

For $k > m$

$$p_k = S^{-1}\prod_{i=0}^{m-1}\frac{\lambda}{\mu(i+1)}\prod_{j=m}^{k-1}\frac{\lambda}{m\mu}$$

$$= S^{-1}\left(\frac{\lambda}{\mu}\right)^k\frac{1}{m!\,m^{k-m}}$$

$$p_k = S^{-1}\frac{\rho^k m^m}{m!} \tag{8.3.3.2}$$

where

$$S = 1 + \frac{\lambda}{\mu} + \frac{\lambda^2}{2!\mu^2} + \ldots + \frac{\lambda^{m-1}}{(m-1)!\,\mu^{m-1}} + \frac{1}{m!}\left(\frac{\lambda}{\mu}\right)^m + \frac{1}{m!}\left(\frac{\lambda}{\mu}\right)^{m+1}\frac{1}{m} + \ldots$$

$$= 1 + \sum_{k=1}^{m-1}\frac{(m\rho)^k}{k!} + \sum_{k=m}^{\infty}\frac{(m\rho)^k}{m!}\frac{1}{m^{k-m}}$$

$$= \left[\sum_{k=0}^{m-1} \frac{(m\rho)^k}{k!} \right] + \left[\frac{m^m}{m!} \sum_{k=m}^{\infty} \rho^k \right]$$

$$= \left[\sum_{k=0}^{m-1} \frac{(m\rho)^k}{k!} \right] + \left[\frac{(m\rho)^m}{m!} \left(\frac{1}{1-\rho} \right) \right] \tag{8.3.3.3}$$

Exercise: Find the probability that an arriving customer is forced to join the queue (Erlang's C formula).

8.3.4. M/M/1/K: Queue with finite storage

We consider a queueing system in which there is a single server, with Poisson arrivals and exponential service time, but we assume the system can hold at most a total of K customers (including the customer in service). Here we have (Figure 8.9)

$$\lambda_k = \lambda \quad k < K$$

$$\lambda_k = 0 \quad k \geq K$$

$$\mu_k = \mu \quad k = 1, 2 \ldots K$$

This implies that customers continue to arrive according to a Poisson process but only those who find the system with less than K customers will be allowed entry otherwise they are considered to be lost and do not return later.

Then we find

$$S = 1 + \frac{\lambda}{\mu} + \frac{(\lambda)}{\mu}^2 + \ldots + \left(\frac{\lambda}{\mu} \right)^K$$

$$= \frac{1 - (\lambda/\mu)^{K+1}}{1 - (\lambda/\mu)} \tag{8.3.4.1}$$

and

$$p_k = S^{-1} \prod_{i=0}^{k-1} \left(\frac{\lambda}{\mu} \right)$$

$$= S^{-1} \left(\frac{\lambda}{\mu} \right)^k$$

$$p_k = \left(\frac{\lambda}{\mu} \right)^k \frac{1 - \left(\frac{\lambda}{\mu} \right)}{1 - \left(\frac{\lambda}{\mu} \right)^{K+1}}, \text{ for } k \leq K \tag{8.3.4.2}$$

$$= 0 \text{ , otherwise}$$

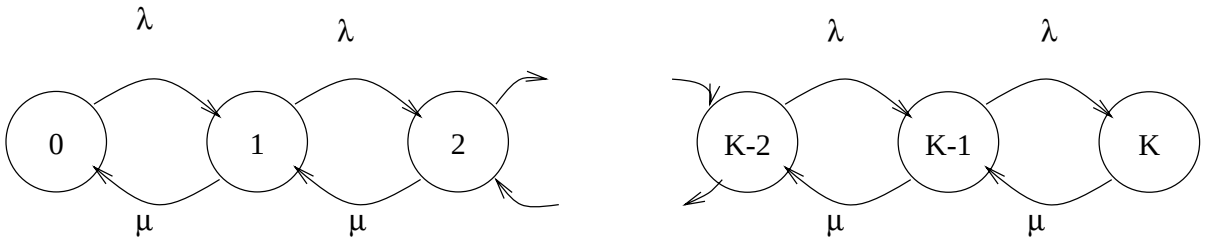


Figure 8.9.

8.3.5. M/M/m/m: m-server loss system

Calls on telephone exchange with m lines. This is the situation which arises when we consider a telephone exchange with just m lines and no facility for holding subscribers who require a line but can not be supplied with one. It is assumed that such calls are lost. We have (see figure 8.10)

$$\lambda_k = \lambda \quad k < m$$

$$\lambda_k = 0 \quad k \geq m$$

$$\mu_k = k\mu \quad k = 1, 2 \ldots m$$

Here we find that

$$S = 1 + \frac{\lambda}{\mu} + \frac{1}{2!}\left(\frac{\lambda}{\mu}\right)^2 + \ldots \frac{1}{m!}\left(\frac{\lambda}{\mu}\right)^m \tag{8.3.5.1}$$

That is

$$p_0 = \left[\sum_{k=0}^{m}\left(\frac{\lambda}{\mu}\right)^k \frac{1}{k!}\right]^{-1}$$

and

$$p_k = S^{-1}\prod_{i=0}^{k-1}\frac{\lambda}{\mu(i+1)} \quad k \leq m$$

$$p_k = \frac{\left(\frac{\lambda}{\mu}\right)^k \frac{1}{k!}}{\sum_{m=0}^{m}\left(\frac{\lambda}{\mu}\right)^k \frac{1}{k!}} \quad k \leq m \tag{8.3.5.2}$$

$$= 0 \qquad k > m$$

In particular, p_m describes the fraction of time that all m servers are busy. The expression for p is called Erlang's loss formula and is given by

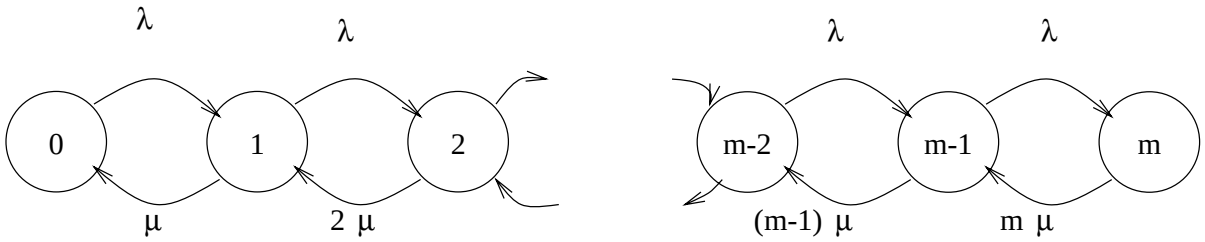


Figure 8.10.

$$p_m = \frac{\left(\frac{\lambda}{\mu}\right)^m \frac{1}{m!}}{\sum_{k=0}^{m}\left(\frac{\lambda}{\mu}\right)^k \frac{1}{k!}} \tag{8.3.5.3}$$

Chapter 9. The Single Server Queue

9.1. The single server queue - M/M/1 queue

A single server queue of packets waiting to be transmitted is shown in figure 9.1.

Assumptions

We assume that the arrivals at the node have a Poisson distribution with average rate of message arrival x. The probability that k messages will arrive in an interval of t seconds is given by

$$p_k(t) = \frac{(\lambda t)^k e^{-\lambda t}}{k!} \quad ; \quad k = 0, 1, 2\ldots$$

The probability distribution function of the interarrival time is

$$A(t) = 1 - e^{-\lambda t} \quad ; \quad t \geq 0$$

We also assume that the length of the messages has an exponential distribution with mean $1/\mu$ bits, i.e.

$$B(x) = 1 - e^{-\mu x} \quad ; \quad x \geq 0$$

We assume that the capacity of the link is C bits per second. Thus the link transmission time or "service time", for a message P bits in length is P/C seconds.

We assume that the stored messages are served in order of their arrival, i.e. a First-in First-out (FIFO) queue discipline.

Let p_k be the equilibrium probability that there are exactly k customers in the system (queue and server). Figure 9.2 shows the states for a singleserver queueing system, with the allowed transitions indicated by arrows.

In equilibrium the probability of finding the system in a given state does not change with time. From Figure 9.2,

$$\lambda p_0 = \mu C p_1$$

$$\lambda p_1 = \mu C p_2$$

$$\lambda p_k = \mu C p_{k+1}$$

From these equations we can solve for p_k in terms of p_0, i.e.

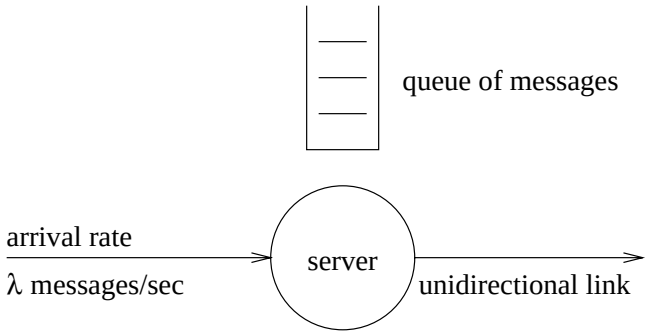


Figure 9.1. A unidirectional link and single-server queue

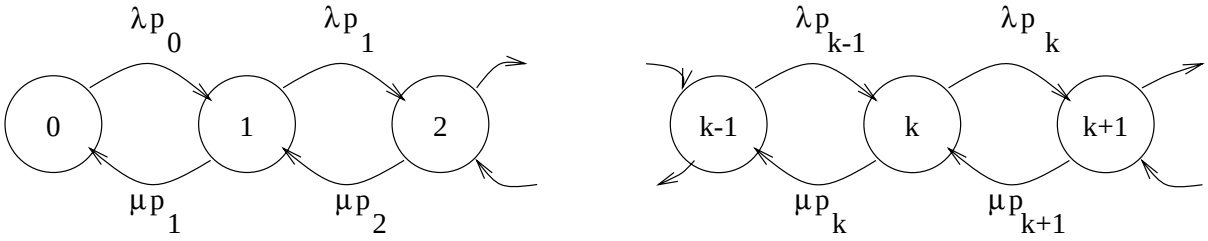


Figure 9.2.

$$p_k = \rho^k p_0 \quad \text{where} \quad \rho = \frac{\lambda}{\mu C}$$

To find p_0 we use the fact that

$$\sum_{k=0}^{\infty} p_k = 0 \quad \rightarrow \quad p_0 = (1-\rho)$$

hence

$$p_k = (1-\rho)\rho^k$$

The mean number of customers in the system, N, can be found directly from the state probabilities

$$N = \sum_{k=0}^{\infty} k p_k$$

$$N = \frac{\rho}{1-\rho}$$

By using Little's theorem, which states that the average number of customers in the system is equal to the average arrival rate of customers to that system, times the average time spent in that system:

$$N = \lambda T$$

We can find the total average time T

$$\boxed{T = \frac{N}{\lambda} = \frac{1}{\mu C(1-\rho)} = \frac{1}{\mu C - \lambda}}$$

The average queueing or waiting time for a message is given by

$$W = T - \text{ service time } = T - \frac{1}{\mu C}$$

$$\boxed{W = \frac{\rho}{\mu C(1-\rho)}}$$

9.2. The M/G/1 queues

The M/G/1 queue is a single-server system with Poisson arrivals and arbitrary service time distribution denoted by $B(x)$. The interarrival time distribution is given by

$$A(t) = 1 - e^{-\lambda t} \quad , \quad t \geq O$$

with an average arrival rate of λ customers per second, a mean interarrival time of $1/\lambda$ sec, and a variance $\sigma_a^2 = 1/\lambda^2$.

The well known formula, the Pollaczek-Khinchin mean value formula, for the average number of customers in an M/G/1 system is given by:

$$N = \rho + \rho^2 \frac{1 + C_b^2}{2(1-\rho)}$$

where

$$C_b \equiv \frac{\sigma_b^2}{\bar{x}^2}$$

From this and by using the Little's theorem we find

$$T = \bar{x} + \frac{\rho\bar{x}(1 + C_b^2)}{2(1-\rho)} \quad ; \quad \rho = \lambda\bar{x}$$

Thus we have the average queueing time

$$W = \frac{\rho\bar{x}(1+C_b^2)}{2(1-\rho)} = \frac{\lambda\overline{x^2}}{2(1-\rho)}$$

9.3. Response time for a contention model

We consider a single multipoint line that connects several terminals to the computer. Messages generated at terminals follow a Poisson distribution. We assume that when a terminal generates a message, it may seize the line, if the line is free, send its message and receive the computer response message; if the line is busy the newly generated message must wait until the line becomes available.

The response time, defined as the time interval between the terminal's seizure of the line and the beginning of the receipt of the response message, is composed of:

(1) Queueing for storage at local terminal

(2) Queueing for lines $\rightarrow W$

(3) Propagation time

(4) Transmission time $\rightarrow t_{tm} + t_{cr}$

(5) Computer processing time $\rightarrow t_{cs}$

We will assume (1) and (3) are negligible. In the context of queueing systems the average service time for a message is given by:

$$\bar{x} = t_{tm} + t_{cs} + t_{cr}$$

If we assume that the service time in general can assume any distribution and only that its average $\bar{x}$ and its variance σ_x^2 are known, then the average queueing delay is given by:

$$W = \frac{\rho\bar{x}\left(1+\dfrac{\sigma_x^2}{\bar{x}^2}\right)}{2(1-\rho)}$$

and the average response time is given by:

$$T = W + \bar{x}$$

Example

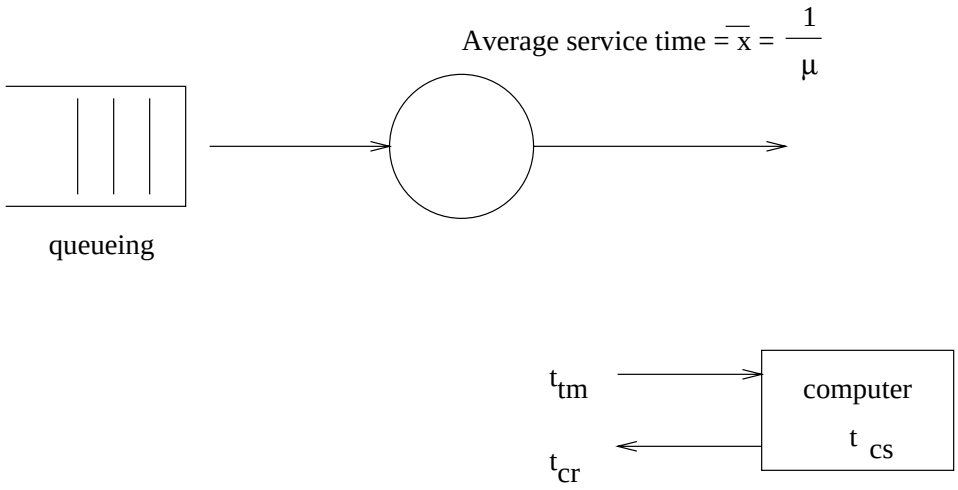


Figure 9.3.

Suppose we have 10 terminals with an average message rate of 90 messages per terminal per hour. Assume all messages have a constant length of 800 bits and the line speed is C = 2400 bps. Assume further that:

$$t_{cs} = 0.2 \text{ sec}$$

$$\sigma_{cs}^2 = 0.1$$

$$t_{cr} = 0.5 \text{ sec}$$

$$\sigma_{cr}^2 = 0.8$$

$$\lambda =$$

$$t_{tm} =$$

$$\bar{x} =$$

$$\sigma_x^2 =$$

$$\rho =$$

And so

$$W = 0.258 \times 1.033 \times \frac{\left[1 + \frac{0.9}{1.033^2}\right]}{2(1-0.258)}$$

$$= 0.331 \text{ sec}$$

and the response time is

$$T = W + \bar{x} = 0.331 + 1.033$$

$$= 1.374 \text{ seconds}$$

The response time is shown graphically in Figure 9.4.

9.4. Networks of M/M/1 queues

The results derived in section 9.2 for the M/M/1 queue can be directly applied to the problem of finding the queueing delay for packets in an IMP. However, when applied to a network of queues we face a number of problems.

The first problem is that the communication channels are not isolated. The output of one channel becomes the input to another. Packets in a single IMP may come from several channels. Thus the input to a certain line is no longer a single external Poisson process but the sum of the outputs of several other network lines. Fortunately, it has been shown by Burke that if the output of several M/M/1 queues feed into the input of another queue, the resulting input arrival process is also a Poisson process, with mean equal to the sum of the means of the feeding process. This is shown in figure 9.5.

Even more fortunate, Jackson has shown that an open network of M/M/1 queues can be analyzed as though each one were isolated from all the others. All that is required is the mean input rate.

Another problem is that once a packet is generated at a node it retains its size throughout its passage through the network. This property introduces correlations into the system and causes severe mathematical difficulties in the analysis. Kleinrock therefore introduced the "Independence Assumption", which assumes that every time a packet arrives at a node, a new length is chosen for it at random from a negative exponential distribution with average length $1/\mu$ bits. This assumption is contradictory to reality but simulations and actual measurements show that it is quite reasonable to make, and allows the model to produce reliable results.

As a consequence of the independence assumption, and of the Burke and Jackson theorems, the queues at the node of packets waiting for links to become available can each be treated independently according to the single server queue results.

As an example, consider the 3-node network of Figure 9.6

$$\lambda_1 = \gamma_{13} + \gamma_{12}$$

$$\lambda_2 = \gamma_{13} + \gamma_{23}$$

$$\lambda_3 = \gamma_{31} + \gamma_{32}$$

$$\lambda_4 = \gamma_{21} + \gamma_{31}$$

The average time in the system for messages that traverse link 1 is

$$T_1 = \frac{1}{\mu C_1 - (\gamma_{13} + \gamma_{12})}$$

for messages that traverse link 2

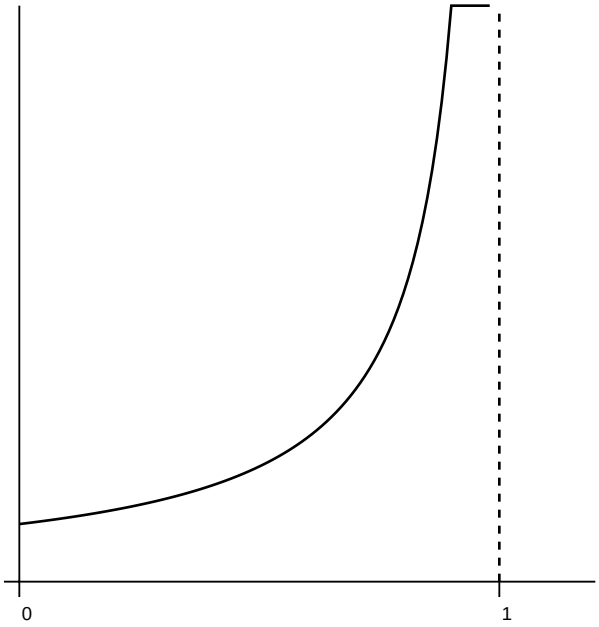


Figure 9.4.

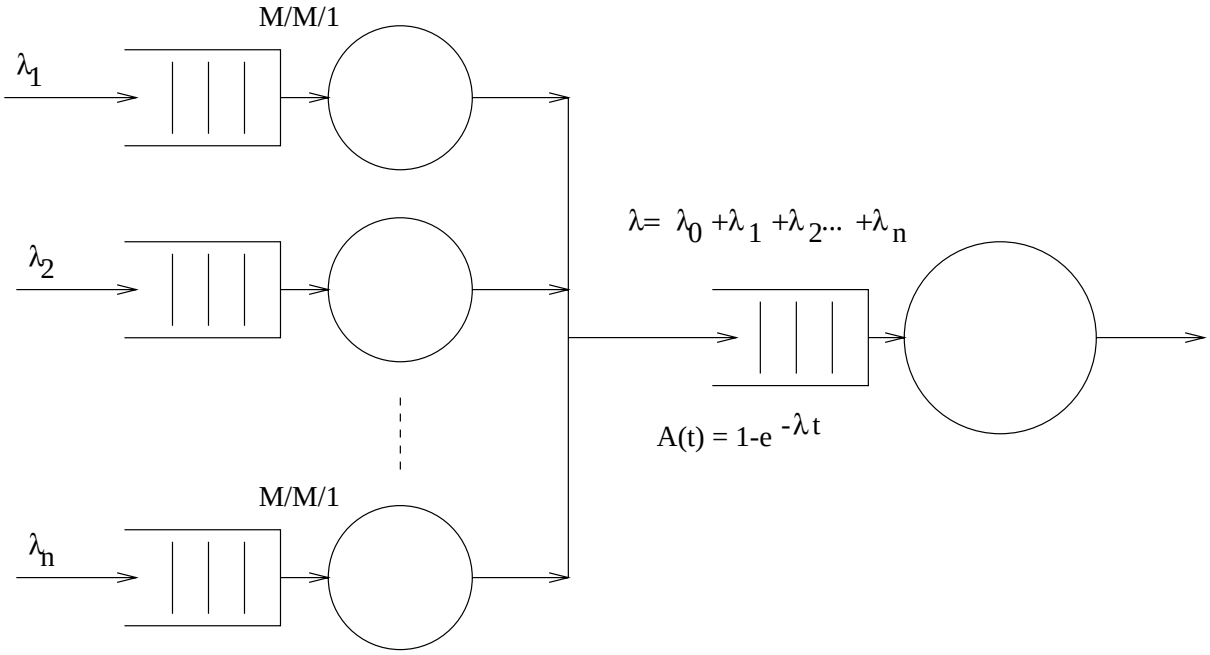


Figure 9.5.

$$T_2 = \frac{1}{\mu C_2 - (\gamma_{13} + \gamma_{23})}$$

and so on. Thus the average time in the system for messages going from source node 1 to destination node 3 is $T_1 + T_2$.

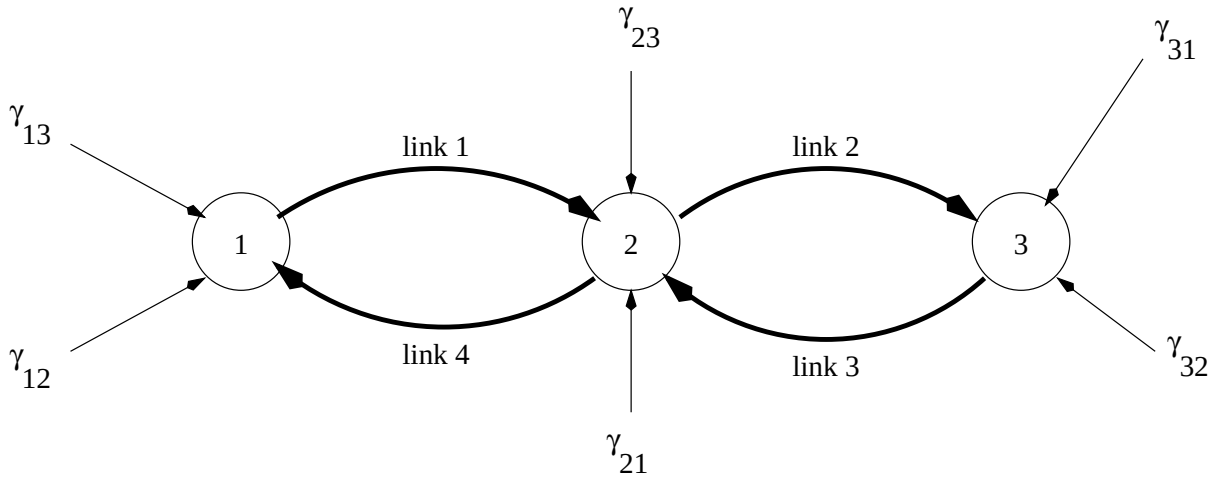


Figure 9.6.

The average time in the system for all packets, (i.e. the average packet delay time through the network) is defined as

$$T = \sum_{i=1}^{m} \frac{\lambda_i T_i}{\gamma}$$

where m is the number of links in the network and

$$\gamma = \sum_{i=1}^{n} \sum_{j=1}^{n} \gamma_{ij}$$

where n is the number of nodes in the network.

Chapter 10. Capacity Assignment in Distributed Networks

10.1. Introduction

Consider a computer communication network with M-channels and N-nodes. The M communication channels are assumed to be noiseless, reliable and to have a capacity denoted by C_i (bits per second) for the ith channel. The N nodes refer to the packet switching centres (IMPS). Traffic entering the network from external sources (i.e. from Hosts) forms a Poisson process with a mean γ_{jk} (packets per second) for those packets originating at node j and destined for node k. The total external traffic entering the network is defined by

$$\gamma = \sum_{j=1}^{N} \sum_{k=1}^{N} \gamma_{jk}$$

Since each channel in the network is considered to be a separate server, γ_i is defined as the average flow (packets per second) in the ith channel. The total traffic within the network is defined by

$$\lambda = \sum_{i=1}^{M} \lambda_i$$

We further assume that the cost of constructing the ith channel with capacity C_i is given by $d_i(C_i)$. The total cost of the network (assuming to consist only of the cost for channel construction) is given by

$$D = \sum_{i=1}^{M} d_i(C_i)$$

The average packet delay defined as the average total time that a message spends in the network is

$$T = E[\text{packet delay}]$$

10.2. Square root channel capacity assignment

The optimization problem here is to assign capacity C_i to each link of the network in such a way that the average packet delay T through the network is minimized subject to a certain constraint.

More precisely we can put it as follows:

CA PROBLEM

Given: Flows $\{\lambda_i\}$ and network topology
Minimize: T
With respect to: $\{C_i\}$
Under constraint: $D = \sum_{i=1}^{M} d_i(C_i)$

Given $\{\lambda_i\}$ on each link means that we must know the routing algorithm. λ_i is then derived from the end-to-end traffic $\{\gamma_{jk}\}$. $d_i(C_i)$ can be a very general function, here we only consider a linear function $d_i C_i$.

SOLUTION

To minimize T we proceed by forming the Lagrangian as follows

$$L = T + \beta[\sum_{i=1}^{M} d_i C_i - D]$$

$$= \sum_{i=1}^{M} \frac{\lambda_i}{\gamma}[\frac{1}{\mu C_i - \lambda i}] + \beta[\sum_{i=1}^{M} d_i C_i - D]$$

As usual we must satisfy the following set of M equations

$$\frac{\partial L}{\partial C_i} = 0 \quad ; \quad i = 1, 2 \ldots M$$

i.e.

$$0 = \frac{\lambda_i}{\gamma} \frac{-\mu}{\left(\mu C_i - \lambda suni\right)^2} + \beta d_i$$

or

$$C_i = \frac{\lambda_i}{\mu} + \left(\frac{\lambda_i}{d_i}\right)^2 \frac{1}{\sqrt{\mu\beta\gamma}}$$

To find β we multiply C_i by d_i and sum over all i:

$$\sum_{i=1}^{M} d_i C_i =$$

$$\frac{1}{\sqrt{\beta\mu\gamma}} =$$

and by defining the "excess dollars" D_e by

$$D_e \equiv D - \sum_{i=1}^{M} \frac{\lambda_i d_i}{\mu}$$

The optimal solution to the linear CA problem is

$$C_i = \frac{\lambda_i}{\mu} + \left(\frac{D_e}{d_i}\right) \frac{\sqrt{\lambda_i d_i}}{\sum_{j=1}^{M} \sqrt{\lambda_j d_j}} \quad , \quad i = 1, 2 \ldots M$$

This assignment of capacity is called the "Square root channel capacity assignment" since C_i has a term proportional to $\sqrt{\lambda_i}$. The minimum average time delay, found by using this optimal capacity assignment is given by

$$T = \sum_{i=2}^{M} \frac{\lambda_i}{\gamma} \left[\frac{1}{\mu C_i - \lambda_i} \right]$$

$$= \frac{\bar{n}}{\mu D_e} \left[\sum_{i=1}^{M} \sqrt{\left(\frac{\lambda_i d_i}{\lambda} \right)} \right]^2$$

where $\bar{n}$ denotes the mean number of hops per packet, i.e.

$$\bar{n} = \frac{\lambda}{\gamma}$$

10.3. A special case

An important special case is when $d_i = d$ (without any loss of generality, we can assume $d = 1$). This case appears when one considers satellite communication channels in which the distance between any two points on earth within the coverage of the satellite is essentially the same regardless of the terrestrial distance between these two points. In this case

$$D = \sum C_i = C$$

i.e. the total capacity is held constant. The optimal assignment is

$$C_i = \frac{\lambda_i}{\mu} + C(1 - \bar{n}\rho^*) \frac{\sqrt{\lambda_i}}{\sum_{j=1}^{M} \sqrt{\lambda_j}} \quad i = 1, 2 \ldots M$$

$$T = \frac{\overline{n}\left(\sum_{i=1}^{M}\sqrt{\frac{\lambda_i}{\lambda}}\right)^2}{\mu C(1-\overline{n}\rho^*)}$$

where

$$\rho^* \equiv \frac{\gamma}{\mu C}$$

Compare these equations with those in Schwartz, where $\rho = \frac{\lambda}{\mu C}$ which is equivalent to traffic intensity parameter for the entire network.

$$C_i = \frac{\lambda_i}{\mu} + C(1-\rho)\frac{\sqrt{\lambda_i}}{\sum_{j=1}^{M}\sqrt{\lambda_j}}$$

$$T = \frac{\left(\sum_{i=1}^{M}\sqrt{\lambda_i}\right)}{\gamma\mu C(1-\rho)}$$

REMARKS

1. From the expression for T it can be seen that T is proportional to $\overline{n}$. This suggests that the minimum delay is obtained with a fully connected network. (See figure 10.1).

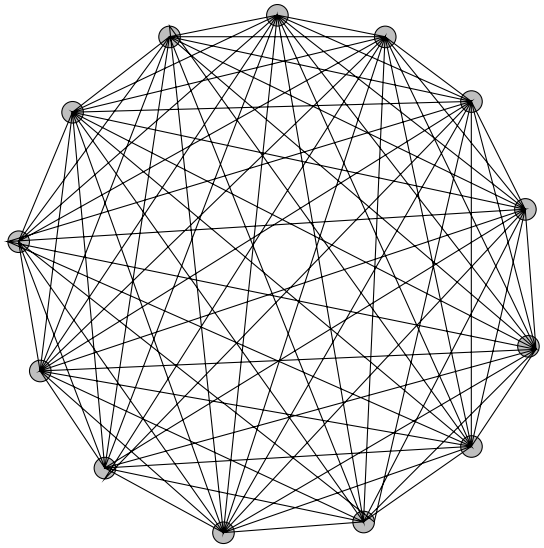
2. On the other hand, the term $\left(\sum\frac{\lambda i}{\lambda}\right)^2$ in the expression for T is minimum if one of these terms is 1 and all the rest are zero. This indicates that T is minimized if traffic is concentrated on few large capacity links and very little traffic on the others. The loop network has this property, but unfortunately the average number of hops per packet $\overline{n}$ is high.

3. A star network achieves a relatively high concentration of traffic on each link and an average $\overline{n} \simeq 2$. This indicates that we should choose a topology which falls between fully connected and star configurations. It is suggested by Kleinrock that for $\rho \to 0$ a star-like net is more appropriate, and when $\rho \to 1$ a more connected net is a better choice.

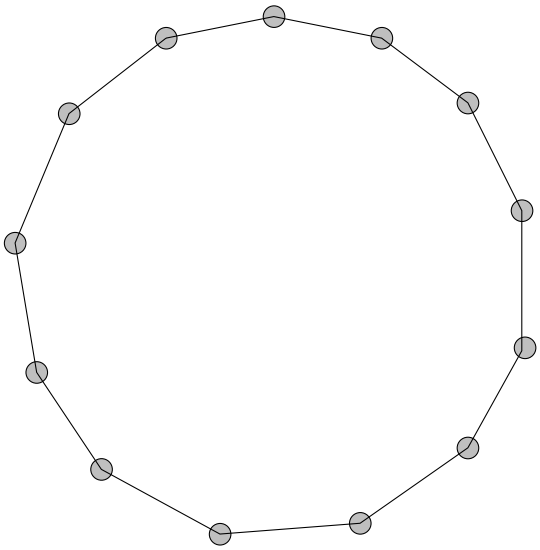
10.4. Some other capacity assignments

1. Uniform assignment in which every line gets an equal amount of capacity, i.e. $C_i = \frac{C}{M}$ for all M.

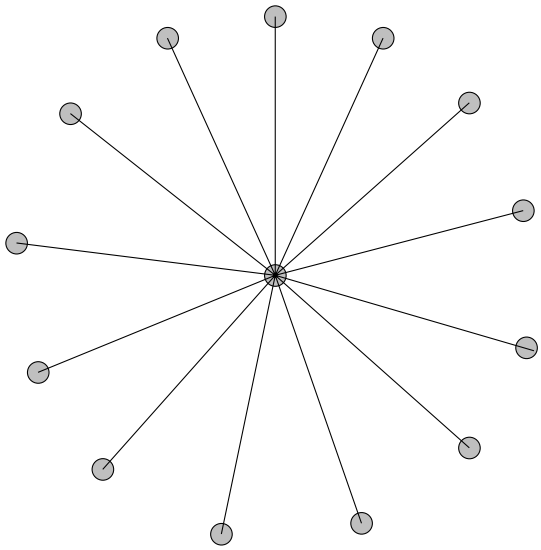
   It will be seen later that the uniform assignment increases the overall average time delay



(a)



(b)



(c)

Figure 10.1. Three topologies of interest: (a) the fully connected net; (b) the loop network; (c) the star net

per packet but reduces the difference in time delays on light and heavy traffic links.

2. Proportional assignment in which C_i is proportional to1< the traffic demand λ_i i.e.

$$C_i = \frac{\lambda_i}{\lambda}$$

It will be seen that the proportional assignment increases the average time delay (relative

to the square-root assignment), and also exagerates the distinction between light and heavy traffic lines. The lighter user is penalized in favor of the heavy user.

3. The CA problem can be generalized as follows:

   The problem is to choose C_i to minimize

$$T^{(k)} = \left[\sum_{i=1}^{M} \frac{\lambda_i}{\lambda} \left(T_i \right)^k \right]^{1/k}$$

   subject to the constraint:

$$D = \sum_{i=1}^{M} d_i C_i$$

   For $k = 1$ we have considered in section 10.1

   For $k = 2$ we have a mean-squared delay to be minimized

   For $k \to \infty$ we have a min-max criterion where the largest time delay on any links is being minimized, subject to the constraint of cost.

   A Lagrange multiplier technique is used as in section 10.1, and the optimum capacity assignment for arbitrary k is given by

$$C_i^{(k)} = \frac{\lambda_i}{\mu} + \frac{D_e}{d_i} \frac{\left(\lambda_i d_i^k \right)^{1/(1+k)}}{\sum_{j=1}^{M} \left(\lambda_i d_i^k \right)^{1/(1+k)}} \quad , \; i = 1, 2 \ldots M$$

   and

$$T^{(k)} = \frac{\left(\bar{n} \right)^{1/k}}{\mu D_e} \left[\sum_{i=1}^{M} \left(\frac{\lambda_i d_i^k}{\lambda} \right)^{1/(1+k)} \right]^{(1+k)/k}$$

10.5. An example

A 7-link network is given in figure 10.2. All links are assumed to be full-duplex and symmetrical in capacity. The traffic reauirement is given in Table 10.1. Assuming that the routing of messages takes on the shortest geoaraphical route as shown in Table 10.2. Assuming further that all messages have the same average length $1/\mu$ bits and that the overall network message capacity in messages/sec is fixed at $\mu C = 192$ messages/sec. Find the capacity of each line so that the average

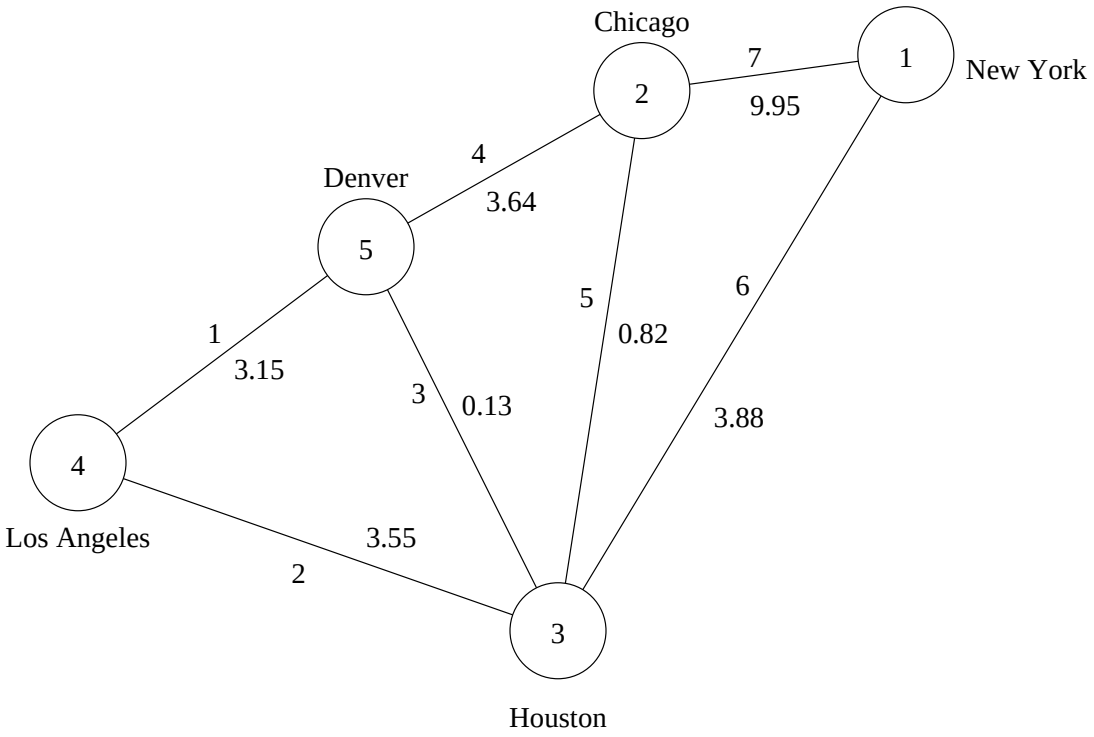


Figure 10.2. Network example (shortest distance routing)

	Destination: New York	Chicago	Houston	Los Angeles	Denver
Source City	1	2	3	4	5
1. New York		9.34	0.935	2.94	0.610
2. Chicago	9.34		0.820	2.40	0.628
3. Houston	0.935	0.820		0.608	0.131
4. Los Angeles	2.94	2.40	0.608		0.753
5. Denver	0.610	0.628	0.131	0.753	

Table 10.1. Traffic Matrix Network

	Destination:				
source	1	2	3	4	5
1		$1-2(l_7)$	$1-3(l_6)$	$1-3-4(l_6,l_2)$	$1-2-5(l_7,l_4)$
2	$2-1(l_7)$		$2-3(l_5)$	$2-5-4(l_4,l_1)$	$2-5(l_4)$
3	$3-1(l_6)$	$3-2(l_5)$		$3-4(l_2)$	$3-5(l_3)$
4	$4-3-1(l_2,l_6)$	$4-5-2(l_1,l_4)$	$4-3(l_2)$		$4-5(l_1)$
5	$5-2-1(l_4,l_7)$	$5-2(l_4)$	$5-3(l_3)$	$5-4(l_1)$	

Table 10.2. Routing

message time delay is minimized. Find the minimum time delay.

Answer

From the traffic requirement γ_{ij}. and the routing assignment, the traffic flow (one way) on each link can be derived as follows:

$$\lambda_1 = \gamma_{24} + \gamma_{45} = 2.40 + 0.753 = 3.15 \text{ messages/sec one way}$$

$$\lambda_2 = \gamma_{14} + \gamma_{34} = 2.94 + 0.608 = 3.55$$

$$\lambda_3 = \gamma_{35} = 0.131$$

$$\lambda_4 = \gamma_{15} + \gamma_{24} + \gamma_{25} = 0.610 + 2.40 + 0.628 = 3.64$$

$$\lambda_5 = \gamma_{23} = 0.820$$

$$\lambda_6 = \gamma_{13} + \gamma_{14} = 0.935 + 2.94 = 3.88$$

$$\lambda_7 = \gamma_{12} + \gamma_{15} = 9.34 + 0.61 = 9.95$$

The total one way link traffic is

$$\lambda = \sum_{i=1}^{7} \lambda_i = 25.12 \text{ messages/sec}$$

The total number of messages/sec entering the entire network on the average is

$$\gamma = \sum_{jk} \gamma_{jk} = 38.3 \text{ messages/sec}$$

For one way traffic $\gamma' = \dfrac{\gamma}{2} = 19.15$ messages/sec.

The average number of hops per packet is

$$\bar{n} = \frac{\lambda}{\gamma'} = \frac{25.12}{19.15} = 1.3$$

The Capacity C_i can be calculated by using the formula

$$C_i = \lambda_i + \mu C(1-\rho)\frac{\sqrt{\lambda_i}}{\sum_{j=1}^{M}\sqrt{\lambda_j}}$$

here $\rho = \dfrac{\lambda}{\mu C} = \dfrac{25.12}{192} = 0.13 \Rightarrow 1-\rho = 0.87$

and so

$$\sum_{j=1}^{M}\sqrt{\lambda_j} = \sqrt{3.15} + \sqrt{3.55} + \sqrt{0.13} + \sqrt{3.64} + \sqrt{0.82} + \sqrt{3.88} + \sqrt{9.95}$$

$$= 1.77 + 1.88 + 0.36 + 1.91 + 0.91 + 1.97 + 3.15$$

$$= 11.95$$

$$\mu C_1 = \lambda_1 + 192 \times 0.87\frac{\sqrt{\lambda_1}}{11.95}$$

$$= \lambda_1 + 13.98\sqrt{\lambda_1}$$

$$\therefore \mu C_1 = 3.15 + 13.98 \times 1.77 \ = 27.89 \approx 28$$

$$\mu C_2 = 3.55 + 13.98 \times 1.88 \ = 29.83 \approx 30$$

$$\mu C_3 = 0.13 + 13.98 \times 0.36 \ = 5.16 \approx 5$$

$$\mu C_4 = 3.64 + 13.98 \times 1.91 \ = 30.34 \approx 30$$

$$\mu C_5 = 0.82 + 13.98 \times 0.91 \ = 13.54 \approx 13.5$$

$$\mu C_6 = 3.88 + 13.98 \times 1.97 \ = 31.42 \approx 31.5$$

$$\mu C_7 = 0.05 + 13.98 \times 3.15 \ = 53.99 \approx 54$$

The minimum average message delay is calculated using the formula

$$T \ = \ \frac{\left[\sum_{i=1}^{M}\sqrt{\lambda_i}\right]^2}{\gamma'\mu C(1-\rho)} \ = \ \frac{11.95^2}{19.15 \times 192 \times 0.87} \ = \ 0.042 \text{ second}$$

The capacities and time delays obtained for using

(i) square root assignment

(ii) equal assignment

(iii) proportional assignment

are summarized in table 10.3.

Exercise

If an alternate routing as shown in Table 10.4 is used, show that the one-way traffic link flows, λ_i are as indicated in figure 10.4, and that

$$\bar{n} = 1.47$$

Link	Demand λ_i, one-way, messages/sec	Square root	T_i (msec)	equal assignment	T_i (msec)	proportional
1	3.15	28	40.4	27.4	41.3	24
2	3.55	30	37.8	27.4	41.9	27.5
3	0.13	5	206	27.4	36.6	1
4	3.64	30	38	27.4	42.1	28
5	0.82	13.5	78.8	27.4	37.6	6.3
6	3.88	31.5	36.2	27.4	42.5	30
7	9.95	54	22.6	27.4	57.3	76.5
		$\overline{T}_{\min} =$ 42msec		$\overline{T}_{=} =$ 57.6msec		$\overline{T}_{prop} =$ 54.8msec

Table 10.3. Capacity allocation μC_i (one-way, messages/sec)

$$T_{\min} = 44\text{msec}$$

	Destination:				
source	1	2	3	4	5
1		l_7	l_6	l_7, l_4, l_1	l_7, l_4
2	l_7		l_5	l_4, l_1	l_4
3	l_6	l_5		l_2	l_3
4	l_1, l_4, l_7	l_1, l_4	l_2		l_1
5	l_4, l_7	l_4	l_3	l_1	

Table 10.4. Alternate Routing

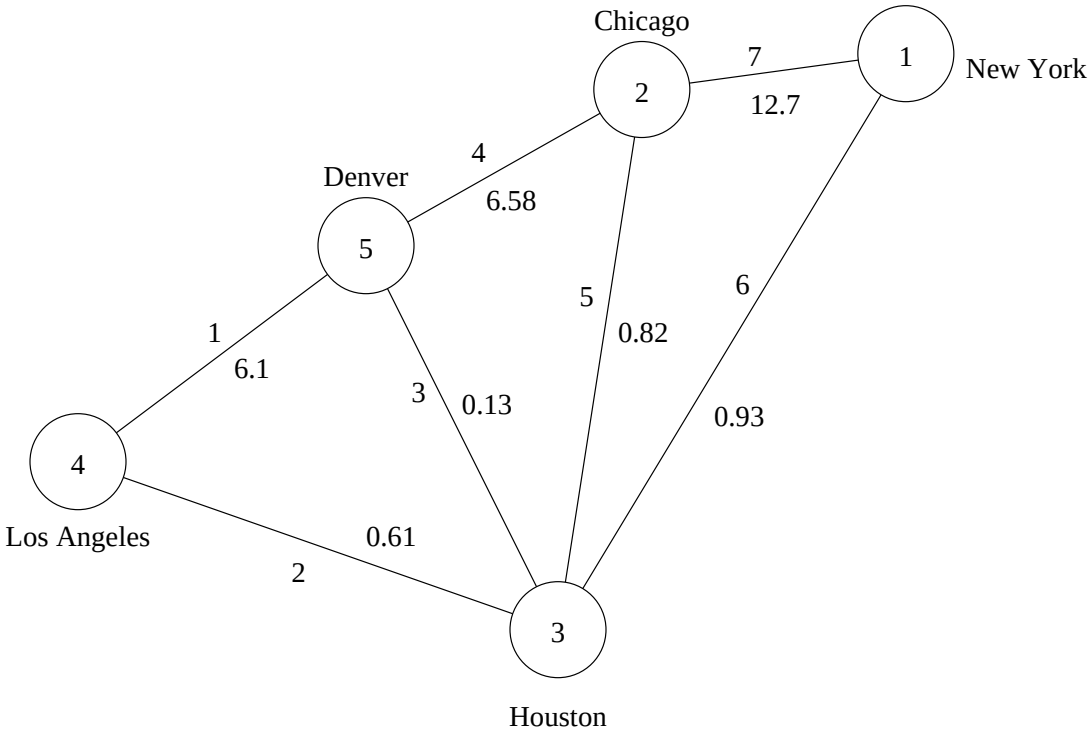


Figure 10.4. Alternate routing, λ_i's (messages/sec) shown, either direction

Chapter 11. Dynamic Buffering and Block Storage

In this chapter we examine the process of dynamic buffering for inbound messages from a number of communication lines to a centralized computer. Static assignment of private storage to each line for message assembly results in costly and inefficient usage of memory resource by ignoring the stochastic behaviour of both message generation and message length. A more efficient assignment would be sharing buffer storage among all the lines by dynamically allocating buffers from a public pool for message assembly.

Messages are generally stored as blocks, with overhead characters assigned to link more than one block corresponding to message together. In this section we develop a stochastic model for the dynamic buffering process. We then use the model to determine the total storage requirement as well as the optimal buffer block size.

11.1. Modelling storage usage

Consider a computer having M communication lines attached. (See figure 11.1). The model could apply equally well to terminals accessing a concentrator in a time-sharing mode, or to concentrators accessing a central processing unit (CPU).

We consider the simplest form of dynamic buffering: when a request for service is made on any line, a block of B characters from the buffer is assigned instantaneously. When the block is filled up another block is in turn assigned instantaneously, the process continues until the message on that particular line has been completely deposited in the buffer. (See figure 11.2).

Each block assigned has B characters, b of which are for message characters, c for the necessary overhead.

The approach here is to assume an unlimited buffer pool and then to find the number of blocks S typically required by the M lines connected such that the probability of exceeding S is some small specified number.

To develop the model we make the following assumptions:

(i) Each of the M lines feeding the computer has identical characteristics, with activity on one line being independent of activity on any other.

(ii) Arrival of individual characters belonging to a single message proceeds at a constant rate (r characters per second) from start of message (SOM) through end of message (EOM).

(iii) Each of the M lines is assumed alternately active and idle. (figure 11.3(a)). During the active period the message is assumed exponentially-distributed with the average message transmission time of $1/\gamma$ sec, this corresponds to an average message length of $1/\mu = r/\gamma$ characters. In the idle interval, the period between the end and start of an active period is

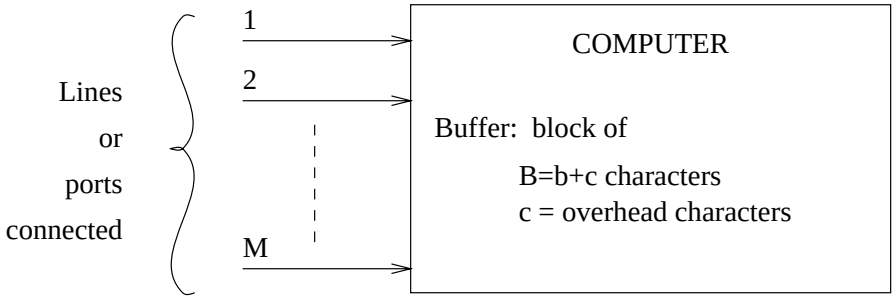
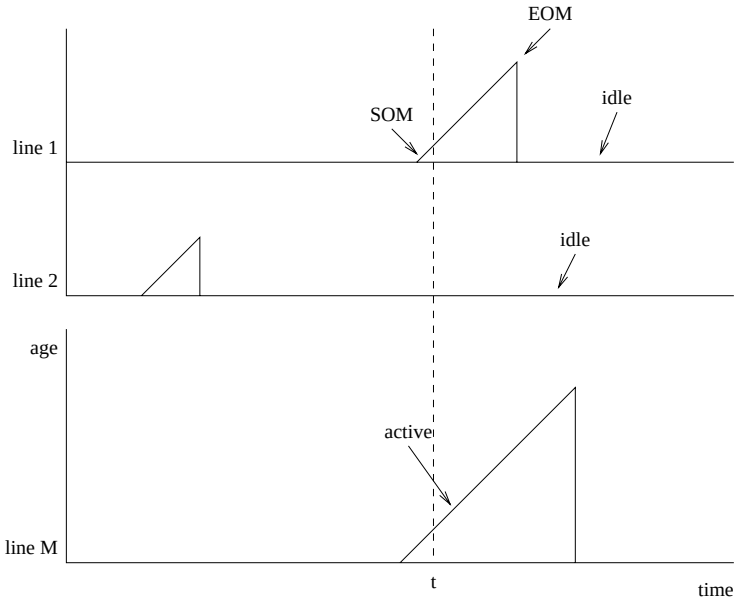
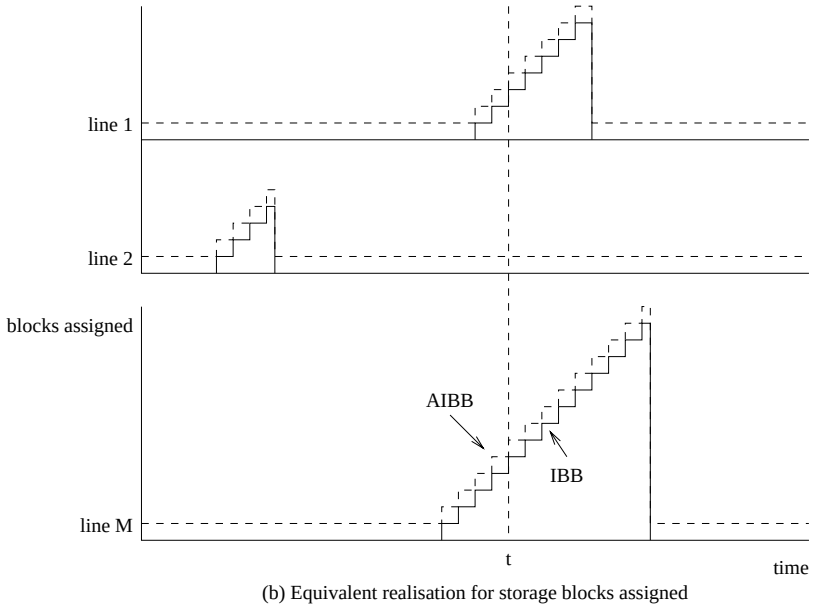


Figure 11.1. *M* lines connected to a computer



(a) Possible realisation for M message times



(b) Equivalent realisation for storage blocks assigned

Figure 11.2. Message age versus storage allocation

also assumed exponentially distributed in length, with average length $1/\lambda$ sec.

With the above assumptions we can model the entire process as consisting of M independent M/M/1 queueing systems as shown in figure 11.3(b).

11.2. Average buffer storage

We now can proceed to find the average number of blocks needed for a typical incoming line.

Since each block contains b message characters, any message of length b or less requires one block. A message with length between b and $2b$ requires 2 blocks, etc. The exponential message length distribution in "time" is sketched in figure 11.4.

Let f_j be the probability that i blocks are needed when a terminal at one of the line requests service. f_j is just the probability that the message length is between $(j-1)\frac{b}{r}$ and $\frac{jb}{r}$ seconds.

$$f_j = \int_{(j-1)\frac{b}{r}}^{\frac{jb}{r}} \gamma e^{-\gamma\tau} d\tau \tag{11.2.1}$$

$$= -e^{-\gamma\tau}\Big|_{(j-1)\frac{b}{r}}^{\frac{jb}{r}} = e^{-\gamma\frac{b}{r}(j-1)}\left[1 - e^{\frac{-\gamma b}{r}}\right]$$

By letting

$$q = e^{\frac{-\gamma b}{r}} = e^{-\mu b}$$

and

$$p = 1 - e^{-\mu b} = 1 - q$$

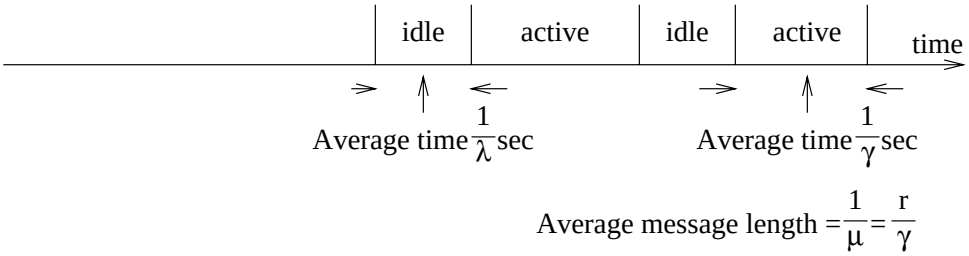
we have

$$f_j = q^{j-1}p \quad ; \quad j = 1, 2\ldots \tag{11.2.2}$$

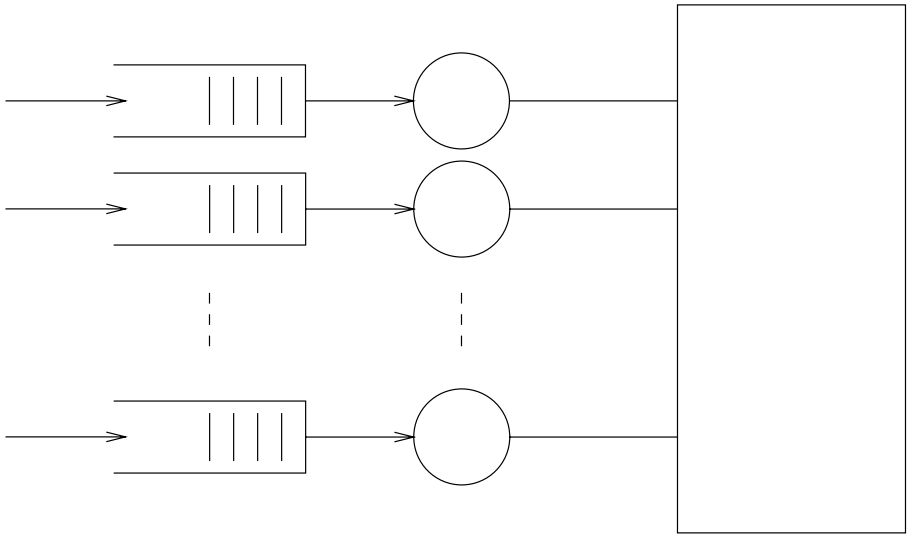
i.e. the number of blocks needed follows the geometric distribution.

Thus the *average* number of blocks needed per active line is

$$E[S_i|\text{ line i active}] = \sum_{j=1}^{\infty} j f_j = p_j \sum_{j=1}^{\infty} j q^{j-1}$$



(a) State of an input line



(b) Equivalent queueing system

Figure 11.3. (a) state of an input line; (b) equivalent queueing system

$$= \frac{p}{(1-q)^2} = \frac{1}{p} = \frac{1}{1-e^{-\mu b}} \tag{11.2.3}$$

The probability that a line is active is given by the fraction of the time the line is active.

$$\text{i.e. } Q = \frac{1/\gamma}{1/\gamma + 1/\lambda} = \frac{\rho}{1+\rho} \;\; ; \text{ where } \rho = \frac{\lambda}{\mu} \tag{11.2.4}$$

The average number of blocks needed by a given line is Q/p. For the M lines connected to the computer, the average number of blocks of buffer storage needed is

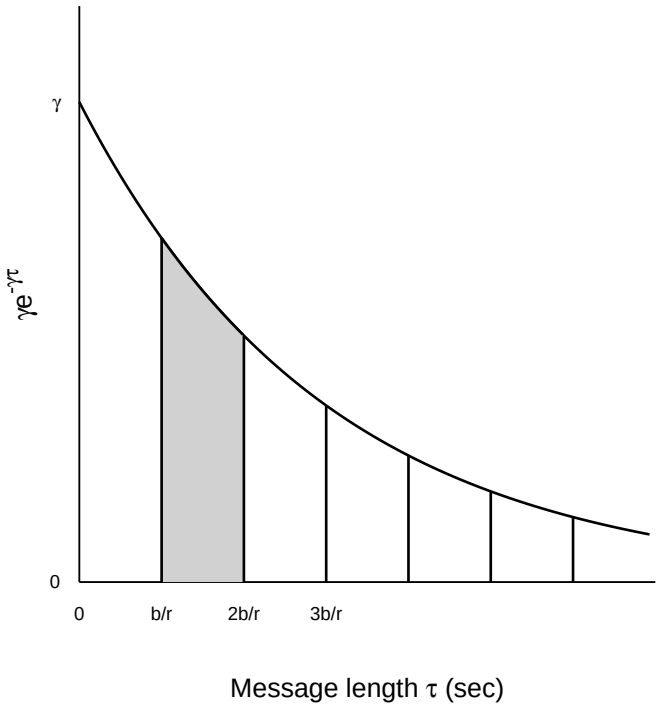


Figure 11.4. Calculation of blocks needed (the shaded region is the probability that 2 blocks are needed)

$$E[S] = \frac{MQ}{p} \tag{11.2.5}$$

The average number of characters needed in the buffer is then

$$E[N] = \tag{11.2.6}$$

Example

A data terminal transmits on average, 3 characters per second, at a rate of $r =$??? characters/sec. If the average message length is $\frac{1}{\mu} =$??? characters, how many blocks of buffer storage are needed for this line if $b = 2$.

Answer

In any time interval the percentage of the active and idle period are:

% active $=$

% idle $=$

The average message transmission time is $\frac{1}{\gamma} = 1$ sec

The average idle period is then $\frac{1}{\lambda} =$

The average number of blocks needed by the line is

$$E[S_i] = \frac{Q}{p} = \left(\frac{0.3}{1 - e^{-20/20}} = \frac{0.3}{0.1} = 3 \right)$$

if $C = 2$ overhead characters are used, an average of 12 characters per incoming line will be needed at the buffer.

From equation 11.2.5 it is clear that there is an optimum block size b that minimizes the number of characters needed. To find b_{opt} we differentiate (11.2.6) with respect to b and set the derivative equal to zero. i.e.

$$\frac{d}{db} E(N) = \frac{d}{db} \left(\frac{MQ(b+c)}{1 - esup - \mu b} \right) = 0 \tag{11.2.7}$$

$$=$$

$$=$$

$$1 + \mu b + \mu c = 1 + \mu b + \frac{(\mu b)^2}{2!} + \frac{(\mu b)^3}{3!} + \ldots \tag{11.2.8}$$

If we assume $\mu C \ll 1$ (i.e. the overhead characters c is small compared to the average message length $1/\mu$) then

$$1 + \mu b + \mu c \simeq 1 + \mu b + \frac{(\mu b)^2}{2}$$

or

$$b_{opt} \simeq \sqrt{2c \left(\frac{1}{\mu} \right)} \tag{11.2.9}$$

Example:

- If $1/\mu = 20$, $c = 2$ then optimum block size is 9 characters. The actual block size is $B = b + c = ll$ characters.
- If $1/\mu = 10$, $c = 3$, $b_{opt} = 8$ characters and a total of 11 characters are still needed for each block.

11.3. Buffer size for a specific probability of overflow

It is found that the average number of characters residing in the buffer is a very poor measure of the buffer sized needed. The buffer size should be chosen such that for given parameters the specified probability of overflow is met. In our case we simply find a storage level whose probability of being exceded matches the overflow criterion.

The method for doing this is straightforward. We have to find the probability h_i that i blocks are required by M lines and then find a level S such that $\Pr[S_L(t) > S] < \varepsilon$ or equivalently $\Pr[S_L(t) \leq S] \geq 1-\varepsilon$, where C is the performance constraint, $L(t)$ denotes a random variable corresponding to the number of active lines. In other words S is the buffer size needed.

The probability h_i is given by:

$$h_i = P[S_L(t) = i] \tag{11.3.1}$$

$$= \sum_{n=0}^{M} \begin{array}{c}\text{Probability that } n \\ \text{out of } M \text{ lines are} \\ \text{active}\end{array} \times \begin{array}{c}\text{Probability that these} \\ n \text{ lines require a total} \\ i \text{ blocks of storage}\end{array}$$

$$= \sum_{n=0}^{M} P[L(t) = n] \times P[X_1(t) + X_2(t) + \ldots X_n(t) = i \mid L(t) = n] \tag{11.3.2}$$

where $X_l(t)$ is a random variable denoting the number of block required by the lth active line.

The probability that n out of M lines are active is just the binomial distribution, with the probability Q (equation 11.2.4) that any one line is active as the basic parameter:

$$P[L(t) = n] = \binom{M}{n} Q^n (1-Q)^{M-n} \quad ; \quad n = 0, 1, 2 \ldots M \tag{11.3.3}$$

The probability $P[X_1 + X_2 + \ldots X_n = i \mid L = n]$ that n active lines require i blocks can be found in terms of the probability f_j. (equation 11.2.2). The following results are from Schultz.

$$h_0 = (1-Q)^M$$

$$h_i = (1-Q)^M q^i \sum_{n=1}^{\min(i,M)} \binom{M}{n} \binom{i-1}{n-1} \left[\frac{Q(1-p)}{q(1-Q)}\right]^n \tag{11.3.4}$$

From which the average buffer size required is again (cf equation 11.2.5)

$$E(S_N) = \frac{MQ}{p}$$

The following curves are taken from Schultz. The average message length is assumed 600 characters, each terminal active 50% of the time, the number of overhead character in each block is 4, and the probability of exceeding the buffer pool size is 10^{-2}.

Exercise

For $M = 1$, $Q = 0.3$, $1/\mu = 20$ characters, $c = 3$ overhead characters, $b = 10$ characters. Show that for a probability of overflow $P[S_L(t) > i] = 10^{-3}$ the buffer storage requirement $s = i = 12$ blocks.

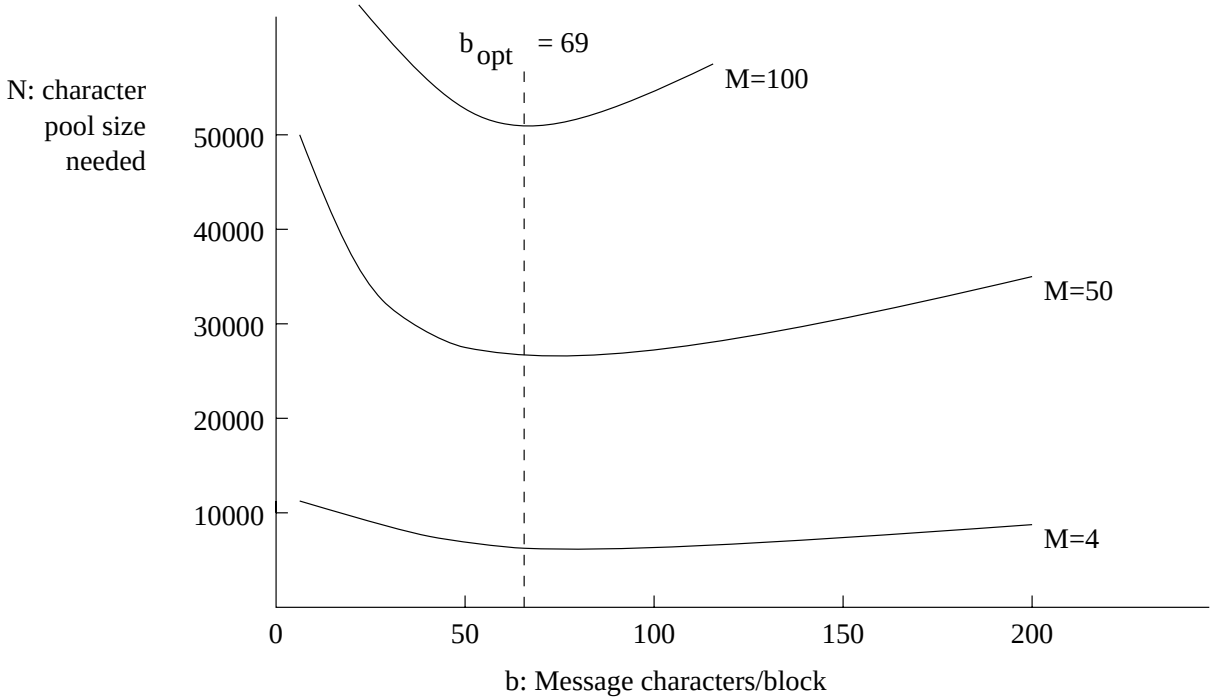


Figure 11.5. Buffer size requirement [approximate]. (From G.D. Schultz, "A Stochastic Model for Message Assembly Buffering with a Comparison of Block Assignment Strategies," *J. of the ACM*, 19, no. 3, July 19721, 483.) This corresponds to $Q = 0.5$, $\frac{1}{\mu} = 600$char., $c = 4$ char., *prob*[$S > i$] $= 10^{-2}$

11.4. Concentration: Finite buffers

In this section we will examine the buffering problems in concentrators for store-and-forward networks. We will consider the effects of finite buffer size, and the probability of buffer overflow using the Poisson message arrival model with geometric lengths.

11.4.1. Finite buffer size model

The model is depicted in figure 11.6. A random number of characters k may arrive every Δsec interval, and a random number n characters are present in the buffer whose maximum size is N. Δ is the character service time. Let π_k be the probability of k characters arriving in Δsec and p_n be the probability that n characters are present in the buffer. The approach used here consists of actually writing down explicit expression for the probabilities of buffer occupancy, and solving the resultant equations recurvively by computer.

At equilibrium the following equations relate the probability of having a particular buffer state present at the end of a Δ second interval to the possible states that could have existed at the end of the previous interval, given no more than 1 character removed in Δsec.

$$p_0 = \pi_0 p_1 + \pi_0 p_0$$

$$p_1 = \pi_0 p_2 + \pi_1 p_1 + \pi_1 p_0$$

$$p_2 = \pi_0 p_3 + \pi_1 p_2 + \pi_2 p_1 + \pi_2 p_0$$

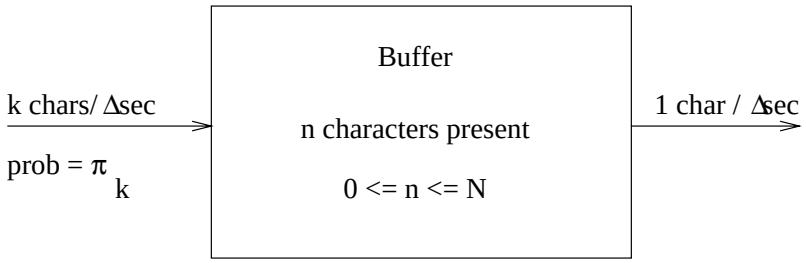


Figure 11.6. Finite buffer model

$$\vdots$$

$$p_n = \pi_0 p_{n+1} + \sum_{i=1}^{n} \pi_{n-i+1} p_i + \pi_n p_0$$

$$\vdots$$

$$p_N = p_N \sum_{i=1}^{\infty} \pi_i + p_{N-1} \sum_{i=2}^{\infty} \pi_i + \ldots + p_2 \sum_{i=N-1}^{\infty} \pi_i + (p_1 + p_0) \sum_{i=N}^{\infty} \pi_i$$

$$= p_N(1-\pi_0) + p_{N-1}(1-\pi_0-\pi_1) + \ldots$$

$$p_{i>N} = 0$$

$$\sum_{i=0}^{N} p_i = 1 \qquad (11.4.1.1)$$

For example, in the second equation of (11.4.1.1), the probability p_1 that one character being present in the buffer consists of 3 events:

(i) Two characters were present during the previous interval, one was emitted, none arrived.

(ii) One was present and left, one arrived.

(iii) The buffer was empty and one arrived.

In the equation for p_N, we assume that when the buffer is full the excess arrived characters are blocked or turned away.

With the input message statistics given, the πs are known explicitly and the p_ns can be found by solving the system of equation (11.4.1.1).

11.4.2. Probability of buffer overflow

For design purposes we would like to find the probability of buffer overflow and relate this to the buffer capacity N and to the traffic intensity. Consider the mode of figure 11.7.

Let λp characters/sec attempt to enter the buffer where X is the Poisson arrival parameter and

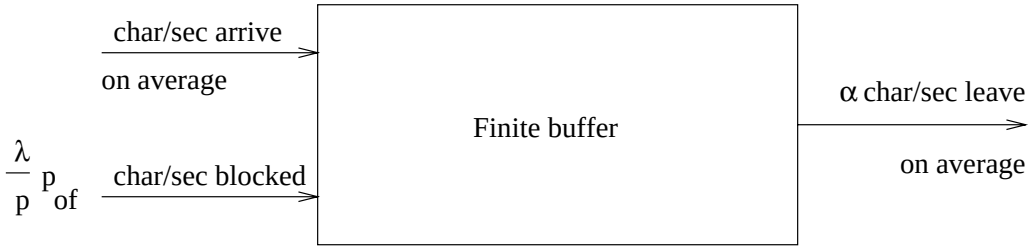


Figure 11.7. Buffer overflow model

$1/p$ is the average length of a message in characters. Let α char/sec represent the average number leaving the buffer. α must differ from the number attempting to enter by exactly those blocked due to the buffer overflow. The fraction of those blocked is just $(\lambda/p)P_{of}$. Hence

$$\alpha\Delta = \left(\frac{\lambda}{p} - \frac{\lambda}{p}P_{of}\right) \tag{11.4.2.1}$$

We can relate α to p_0, the probability the buffer is empty, by the following argument:

- Characters can only leave when the buffer has at least one character stored or is not empty.
- Only one character can leave in Δsec.
- The probability that the buffer is not empty is: $1-p_0$. Hence the average number of character leaving in Δsec is $1\times(1-p_0)$, i.e.

$$\alpha\Delta = 1-p_0 \tag{11.4.2.2}$$

From (11.4.1.1) and (11.4.2.1) the buffer overflow probability is given by

$$P_{of} = 1-\frac{p(1-p_0)}{\lambda\Delta} \tag{11.4.2.3}$$

where p_0 is found by solving the set of equations (11.4.1.1).

Following results are taken from Chu.

The buffer size as a function of message length $1/p$ is sketched in figure 11.8 for overflow probability $P_{of} = 10^{-6}$.

Exercise

Show that for a finite M/M/1 buffer which can accommodate a maximum of M messages, the probability of message blocking and the overflow probability are the same, i.e.

Average message length (characters) $\frac{1}{p}$	P_{of}	$\rho = 0.6$ buffer capacity N (characters)	
		Finite	Exponential
1	10^{-3}	5	12
1	10^{-7}	16	30
2	10^{-3}	22	24
2	10^{-8}	63	69
4	10^{-3}	50	47
4	10^{-8}	153	144
10	10^{-3}	150	118
10	10^{-8}	410	393

Table 11.1. Buffer capacity and overflow probability

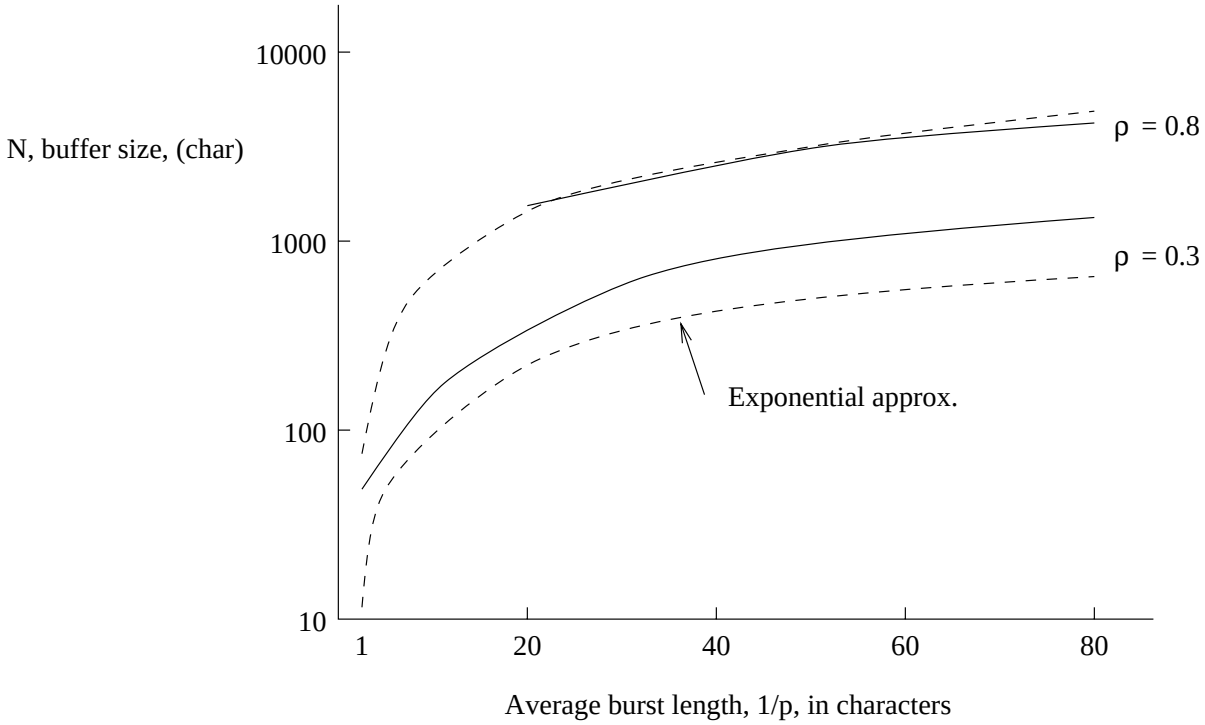


Figure 11.8. Buffer size vs. message length, $P_{of} = 10^{-6}$. (From W.W. Chu, "Buffer Behavior for Batch Poisson Arrivals and Single Constant Output," *IEEE Trans. on Communication Technology*, COM-18, no. 5, Oct 1970, 613-18.)

$$p_{of} = p_B = p_M = \frac{(1-\rho)\rho^M}{1-\rho^{M+1}}$$

The character capacity of an M/M/1 buffer is approximated by

$$N = \frac{M}{\mu}$$

where $1/\mu$ is the average message length in characters.

11.4.3. Example

The above results can be used for some simple design calculations. As an example, say the central processor (CPU) ultimately connects with 46 data terminals as shown in figure 11.9.

Assume that each terminal transmits, at a Poisson rate, an average of 1 message every 2.84 sec. The CPU replies at the same rate so that the number of Poisson-distributed message leaving the computer is 46λ. Assume the CPU messages to each terminal are geometrically distributed in length, with a mean length $1/p = 20$ characters. Say 20% is added for overhead, so that the average message or burst length is 24 characters. If a 480 char/sec trunk is used leaving the CPU what size buffer is needed at the CPU to accommodate messages outbound over this trunk for (a) $p_{of} = 10^3$, (b) $p_{of} = 10^8$?

What if the trunk capacity is now 960 char/sec?

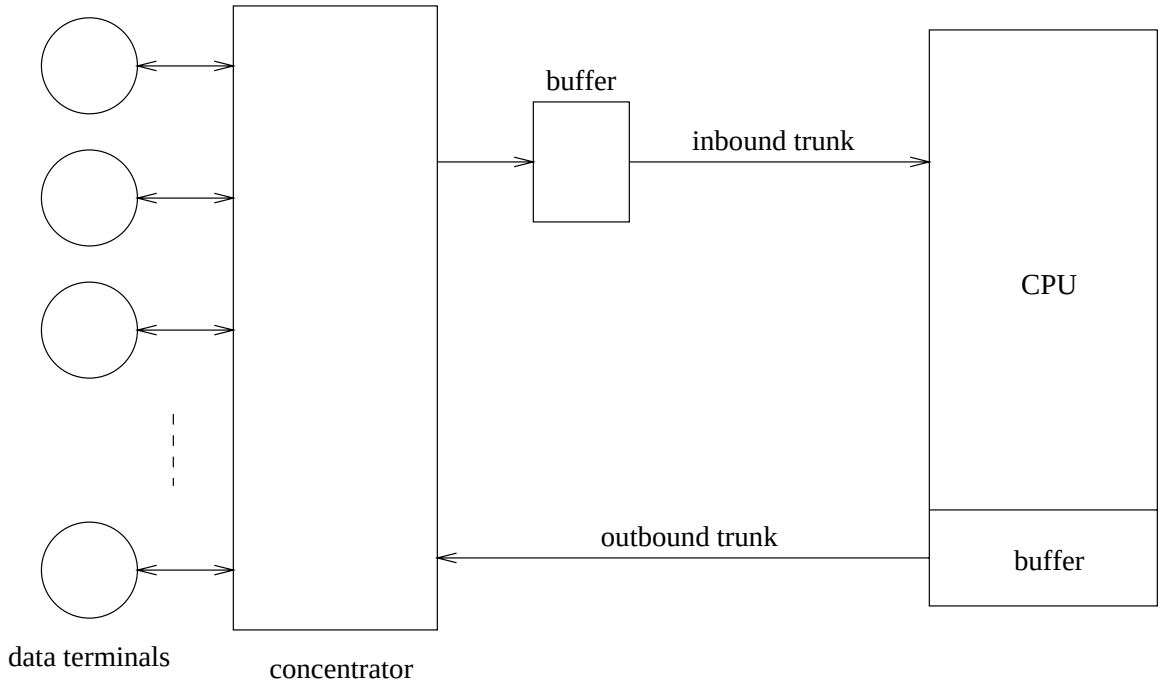


Figure 11.9. Model of rudimentary network (after W.W. Chu, "Buffer behavior for batch Poisson Arrivals")

Chapter 12. Reliability of Networks

One of the requirements usually imposed on computer networks is that they be reliable, that is the networks continue to function properly (relative to certain performance criteria) even if some of its components (nodes, links) fail. The measures of reliability falls into two classes: Deterministic measures and Probabilistic measures. The deterministic measures depend only on the topology of the network. The probabilistic measures depend not only on the network topology but also on the probabilities of failure of network components. In this chapter we describe some of the abovementioned measures.

12.1. Deterministic measures

Network analysis relies heavily on graph theory, so we begin by defining some notations and stating relevant results from graph theory.

12.1.1. Flow in networks

Consider a network or graph $G = [N, L]$ with N nodes and L links. Each link (i, j) from node i to node j has associated with it a link flow f_{ij} and a link capacity c_{ij}. A feasible flow from a source node s to a destination node t is a set of link flows that satisfies the flow conservation equations:

$$\sum_{x \in N} f_{ix} - \sum_{y \in N} f_{yi} = \begin{cases} \gamma_{st} & \text{if } i = s \\ 0 & \text{if } i \neq s \text{ or } t \\ -\gamma_{st} & \text{if } i = t \end{cases} \tag{12.1.1.1}$$

$$0 \geq f_{ij} \leq c_{ij} \quad \text{all } i,j \in N \tag{12.1.1.2}$$

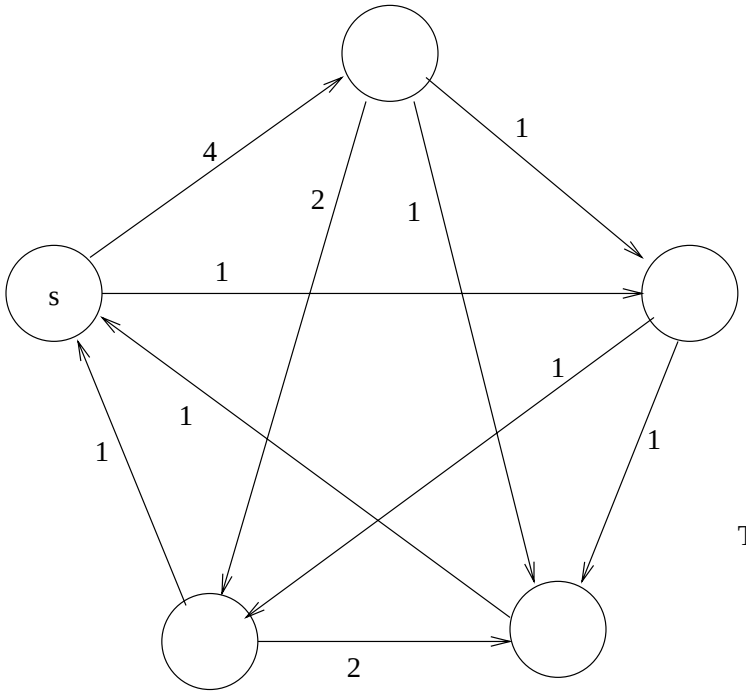
That is, for all nodes other than the source and the destination the total flow into the node equals to the total flow out of that node. γ_{st} is the value of the flow s, t.

An example is shown in figure 12.1.

12.1.2. Cuts

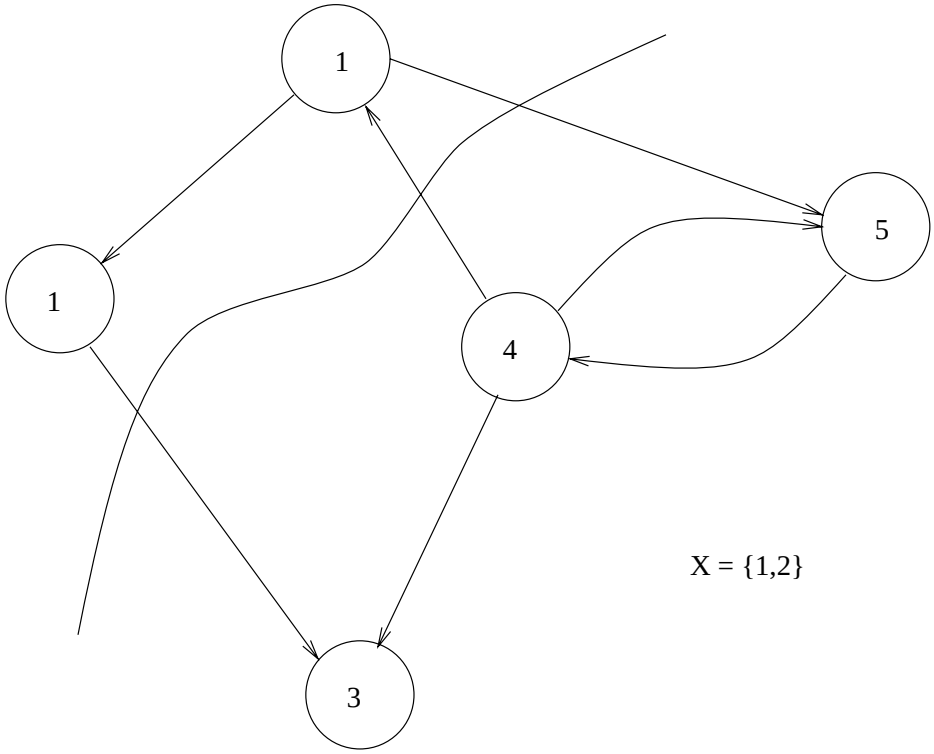
A cut is a set of links that connects one group of nodes (x) to all remaining nodes $(\overline{x})$. A $s-t$ cut separating s and t is a set of links $(x, \bar{x})$ where $s \in x$, $t \in \bar{x}$. For example, the set of links $\{(1,4), (1, 5), (2,3)\}$ is a cut $(x, \bar{x})$ with $x = \{1, 2\}$ as shown in figure 12.2.

A directed i, j link cut-set is a set of links that has the same property as an i, j cut, in addition it is minimal. For example, a cut $(x, \bar{x})$ above is also a $(2, 3)$ cut, but it is not a $2, 3$ link cut-set since the removal of the smaller set of links $\{(2, 3)\}$ breaks every directed route between node 2 and node 3. The set $\{(2, 3)\}$ is a link cut-set.



The value of this flow is 3

Figure 12.1.



X = {1,2}

Figure 12.2.

The capacity of a cut is the sum of the capacities of the links in the cut. A cut with the minimum

capacity is called a minimum cut.

If the source and the destination are in different groups of nodes, all flow from one to the other must pass through the cut that connects them. Therefore, the maximum flow from source to destination cannot exceed the capacity of the cut.

12.1.3. The max-flow min-cut theorem [Ford & Fulkerson]

The maximum flow from source to destination is equal to the capacity of the minimum cut between them.

There are few algorithms for finding out the maximum flow between two nodes, we report below a Labelling algorithm by Ford and Fulkerson. The algorithm also finds which flow pattern achieves this maximum flow.

F-F algorithm:

Initially, set $f_{ij} = 0$ for all (i,j) in L; the algorithm breaks into two parts:

Labelling routine:

A node is considered to be in one of three states: unlabelled, labelled and scanned, or labelled and unscanned. Initially all nodes are unlabelled.

1. Label the source s by $[s, +, e(s) = \infty]$. s is now labelled and unscanned. Goto step 2.

2. Select any labelled and unscanned node x.

   (a) For any unlabelled y such that $f_{xy} < c_{xy}$, label y by $[x, t, e(y)]$ where $e(y) = \min[e(x), c_{xy} - f_{xy}]$. y is now labelled and unscanned.

   (b) For any unlabelled y such that $f_{yx} > 0$, then label y by $[x, -, e(y)]$ where $e(y) = \min[e(x), f_{yx}]$. y is now labelled and unscanned.

   Repeat (a) and (b) until no more y qualify; then change the label on x by encircling its $+$ or $-$ entry. x is now labelled and scanned. Goto step 3.

3. Repeat step 2 until t is labelled (in which case, proceed to the Augmentation Routine) or until no more labels can be assigned. In the latter case the algorithm terminates and the flow currently in the network represents a maximum flow pattern from s to t.

Augmentation Routine

1. Let $z = t$ and go to step 2.

2. If the label on z is $[q, +, e(t)]$ then increase f_{qz} by $e(t)$. If the label on z is $[q, -, e(t)]$ then decrease f_{zq} by $e(t)$. Go to step 3.

3. If $q = s$, erase all labels and return to step 1 of the Labelling Routine. Otherwise, let $z = q$ and return to step 2 of the Augmentation Routine.

12.2. Cohesion or link connectivity

One of the two basic measures of network reliability is cohesion or link connectivity . It is defined as the minimum number of links, c_a, that must be removed from the network $G(N,L)$ to break all routes between at least one pair of nodes.

For example, in figure 12.3, the $A-F$ link connectivity is 2, the $A-C$ link connectivity is 3, and the cohesion of the network is 2.

The link connectivity can be defined in terms of link-disjoint paths. Two paths are said to be link-disjoint if they have no links in common. For example, paths *ACF* and *ABCDEF* in figure 12.4 are link-disjoint. It can be seen that if there are k link-disjoint paths between X and Y, then at least k links must be removed from the graph to disconnect the nodes.

12.2.1. Finding the Cohesion of a Directed Graph

The max-flow min-cut theorem can be used to calculate the number of link-disjoint paths between a given pair of nodes. A modified Labelling algorithm is as follows:

First replace the capacity on the links, if any, with a unit capacity on each link. Now compute the maximum flow between the given nodes, used as source and destination. The resulting flow, which is equal to the number of links in the maximum cut, is also equal to the number of disjoint paths between the nodes.

The cohesion of the network can be calculated in a straightforward way. Just calculate the link-connectivity for each pair of nodes and take the minimum value found.

12.2.2. Finding the Cohesion in an Undirected Graph

Starting with undirected graph $(G'(N,L'))$, we find the link connectivity between s,t as follows:

First, we derive from $G'(N,L)$ a directed graph $G(N,L)$ by:

(a) replacing each link that ends at s in L' by a directed link in L from s to the appropriate node;

(b) replacing each link that ends at t in L' by a directed link in L from the appropriate node to t;

(c) replacing each other link (i,j) in L' by 2 directed links (i,j) and (j,i) in L.

The transformation is illustrated in figure 12.4.

The labelling algorithm can now be applied to find the link connectivity between s,t as before.

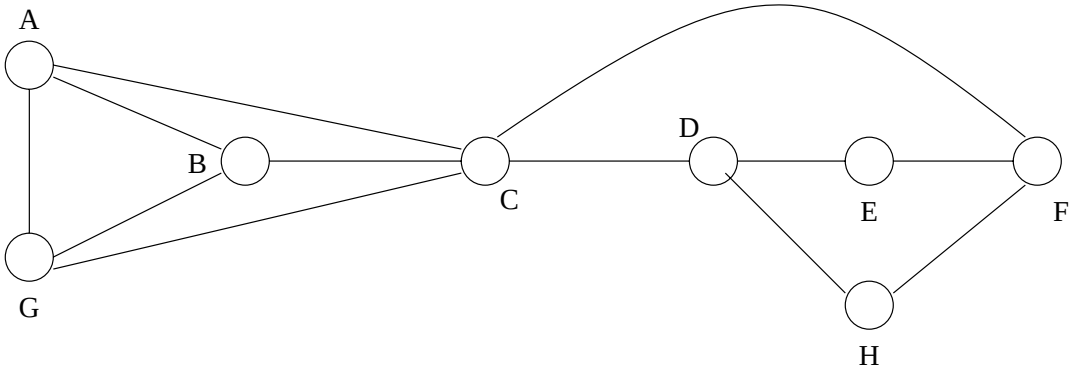


Figure 12.3.

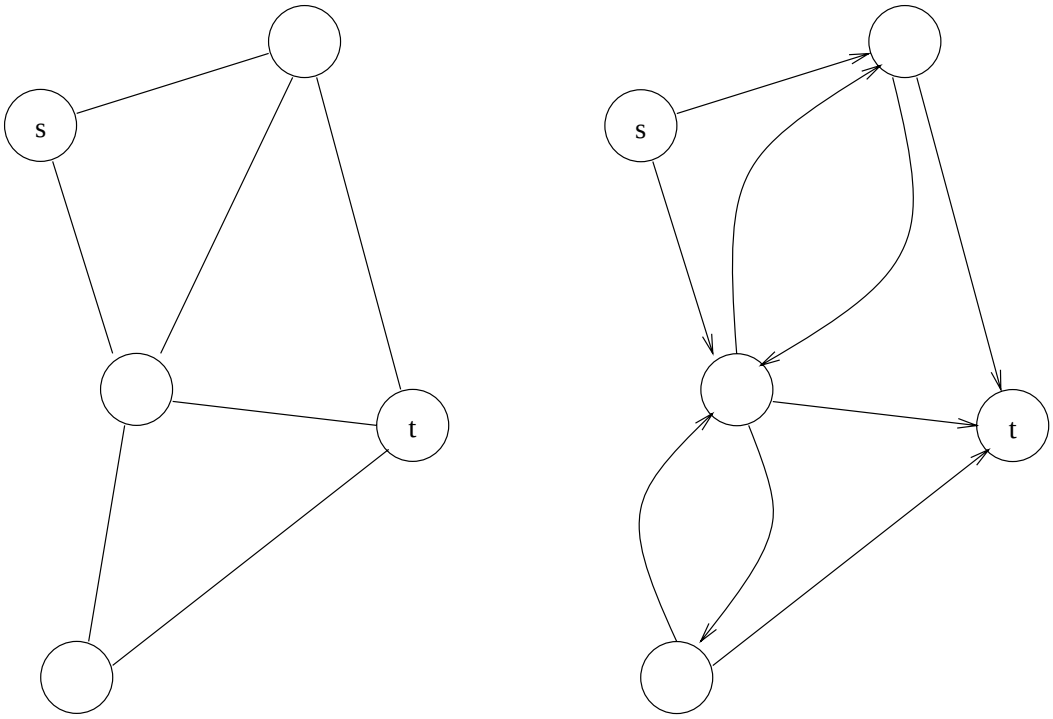


Figure 12.4.

12.3. Node connectivity

The other measure of network reliability is the node connectivity, or just connectivity. It is defined as the minimum number of nodes, c_n, whose removal completely disconnect at least one pair of nodes.

The node connectivity between 2 nodes can be defined in terms of node-disjoint paths. For example there are 3 node disjoint paths in the graph of figure 12.5.

It can be seen that if there are n node disjoint paths between X and Y then at least n nodes must be removed from the graph to disconnect the nodes

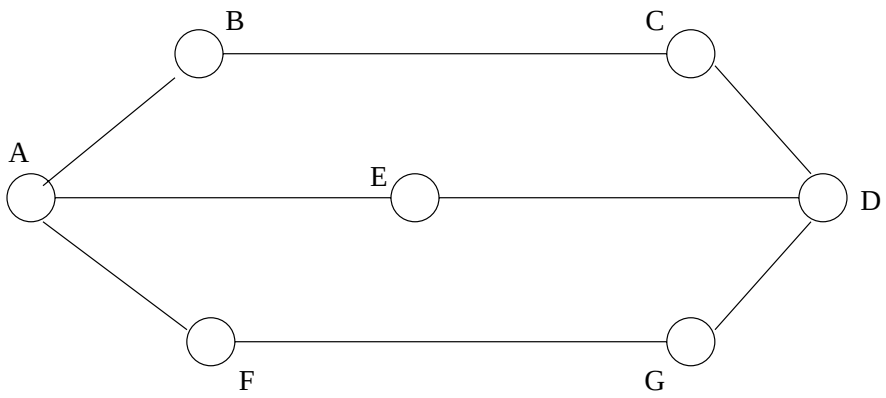


Figure 12.5.

12.3.1. Finding the node connectivity of a undirected graph

The max-flow algorithm can be used again. The basic idea is to convert the problem of finding node-disjoint paths into an equivalent problem involving link-disjoint paths.

The transformation proceeds as follows: take the original (undirected) graph, with its N nodes and L links, convert it into a directed graph with $2N$ nodes and $2L + N$ links by replacing a link between each pair of nodes by a set of links as illustrated in figure 12.6. An example is in figure 12.7.

All that remains to find the number of node-disjoint paths between the source and destination is to run the max-flow algorithm.

The node connectivity of the whole network is found by applying the modified max-flow algorithm to every pair of nodes in the modified network and taking the minimum over all pairs of nodes. For a network with n nodes, $n(n-1)/2$ applications of the max-flow algorithm are needed. In large networks this much computation is infeasible. To find out whether a proposed network is k link-connected or k node-connected more efficient methods can be used. We consider only the case of node connectivity, because node connectivity is stronger requirement than link connectivity. That is, if a graph can withstand the loss of $k + 1$ nodes, it can also withstand the loss of $k + 1$ links, by Whitney's theorem which states that

$$c_n \leq c_a \leq d$$

where d is the minimum degree taken over all nodes in the graph.

12.3.2. Kleitman's Algorithm:

An algorithm due to Kleitman (1969) is as follows:

Choose any node and call it N_1. Verify that the node connectivity between N_1 and every other node in the graph is at least k.

Now delete node N_1 and all its attached links from the graph and choose another node and call it N_2. Verify that this node has at least a node connectivity of $k-1$ with every other node. Next,

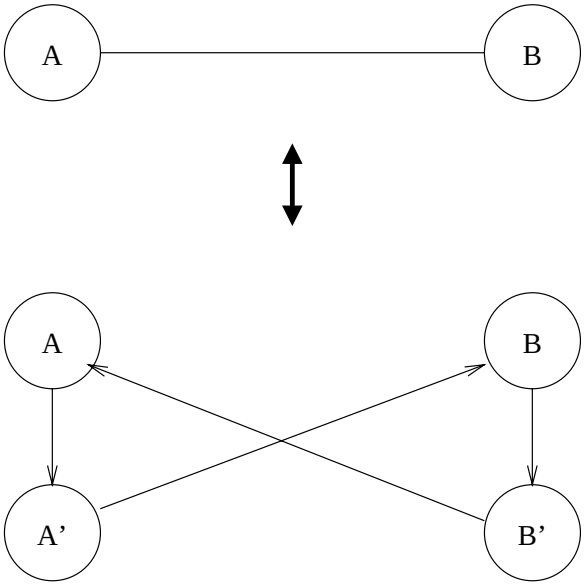


Figure 12.6.

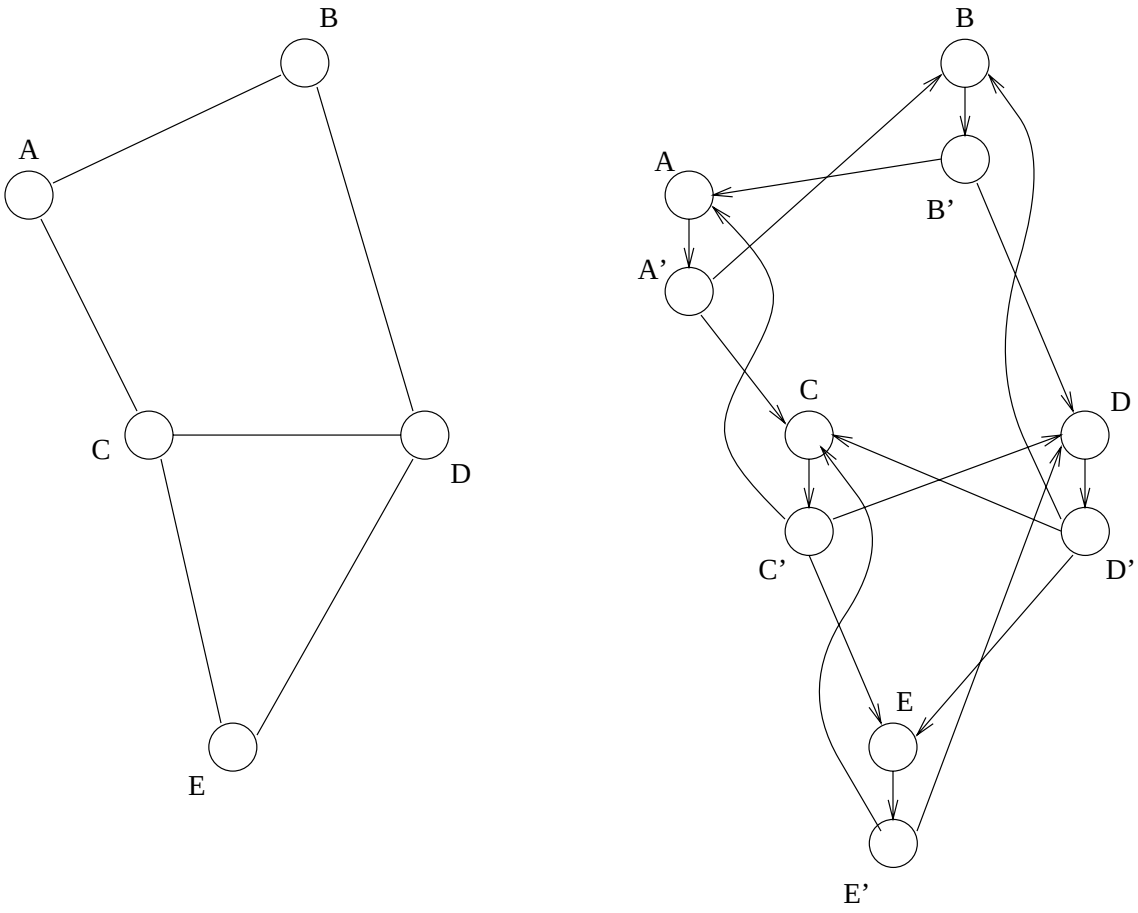


Figure 12.7.

remove N_2 and its attached links from the graph, and choose a third node N_3. Verify that N_3 has at least a node connectivity of $k-2$ with each of the remaining nodes. Continue this process until you have verified that some node N_k is at least 1-connected to all nodes of the remaining graph. If, on the other hand, G is not k-connected, i.e. $c_n < k$, the iterative process will fail at one step and the algorithm need not be carried beyond this step.

An example is illustrated in figure 12.8, showing the original graph is 4-connected.

12.3.3. Even's algorithm

Even (1975) has devised another way to check for connectivity k. Number the nodes $1, 2, \ldots N$ and proceed as follows:

(a) Form the subset consisting of the nodes $1, 2, \ldots k$. Check to see that each node in the subset has k node-disjoint paths to each of the other node. If this step succeeds, continue with step (b); otherwise, stop: the graph obviously does not have k node disjoint paths between all pairs of nodes.

(b) For each node, $j, k < j \leq N$, perform the following operations:

(i) Form the subset $s = \{1, 2, \ldots, j-l\}$

(ii) Add a new node x and connect it to each node in s.

(iii) Verify that there are k node disjoint paths between j and x.

If for some j step (b) fails, the graph does not have k node-disjoint paths between all pairs of nodes; otherwise, it does.

Exercise

Is the graph of figure 12.9 3-connected?

12.4. Probabilistic link and node failures

The links and nodes of the network may fail to operate because of equipment breakdowns, natural disasters, etc... We will assume that each link (i, j) has a probability of failure p_{ij}, that each node i has a probability of failure p_i and that all the link and node failures are statistically independent.

We will calculate for a particular node-pair $s-t$ the probability R_{st} that at least one operable route exists from s to t. R_{st} can be found as follows:

Suppose that there are N possible routes from s to t. Let $E_i, i = 1, \ldots, N$ be the event that route i between s and t is operable; E_iE_j the joint event that routes i and j are both operable, and so forth for events $E_iE_jE_k$, and so on; in each case we assume $i < j < k \ldots \leq N$ so that each combination of routes has a unique representation.

Similarly, let P_i be the probability of event E_i, P_{ij} the probability of event E_{ij}, and so on. We wish to compute the jprobability of the event that at least one among the E_k occurs, i.e. the event

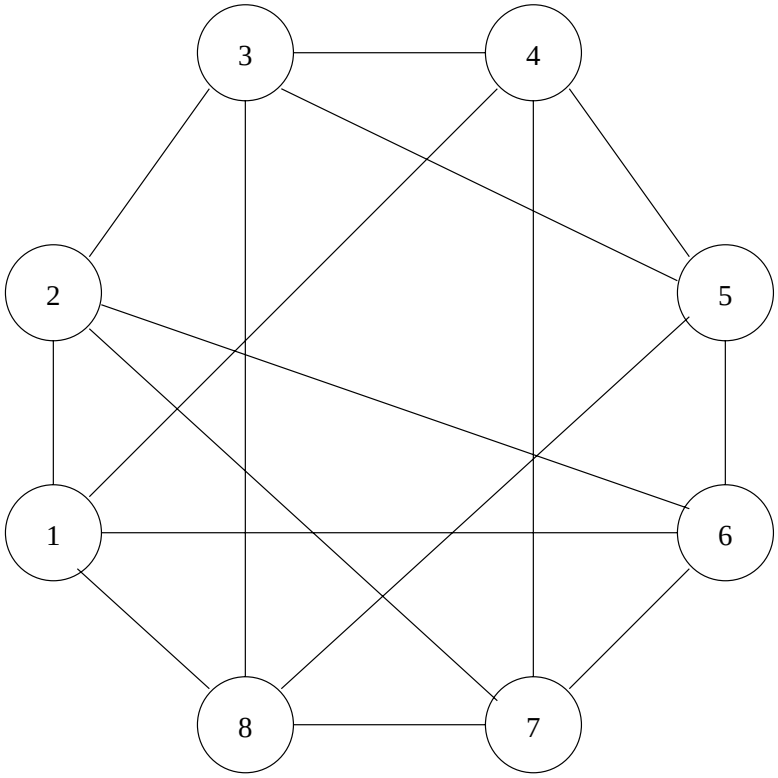


Figure 12.8.

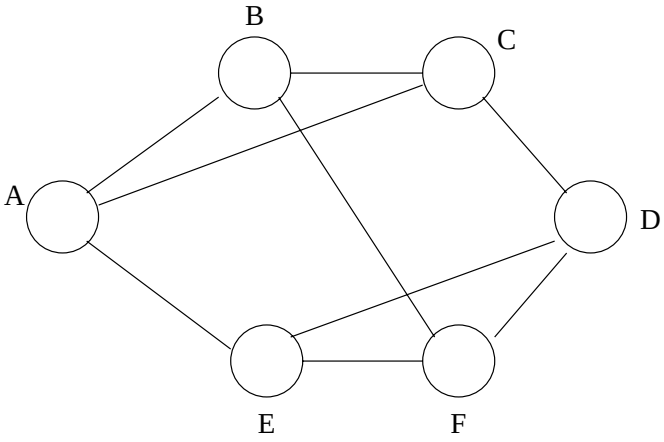


Figure 12.9.

$$E = E_1 \cup E_2 \ldots \cup E_N$$

For our purpose it is not sufficient to know the probabilities of the individual events E_k, but we must be given complete information concerning all possible overlaps, i. e. P_{ij}, P_{ijk}, ...

We define

$$S_1 = \sum P_i, \quad S_2 = \sum P_{ij}, \quad S_3 = \sum P_{ijk}, \quad \ldots$$

We then have

$$\boxed{R_{st} = S_1 - S_2 + S_3 - \ldots \pm S_N}$$

Each term S_k in the above equation is the sum of (k) distinct probabilities for a given k routes out of N being operable.

Example

For the network of figure 12.10 we want to find R_{st}.

Possible routes from s to t are

1. Route 1 $(s,x),(x,t)$; P_1?
2. Route 2 $(s,x),(x,y),(y,t)$; P_2?
3. Route 3 $(s,y),(y,x),(x,t)$; P_3?
4. Route 4 $(s,y),(y,t)$; P_4?

$$S_1 = P_1 + P_2 + P_3 + P_4$$

$$S_2 = P_{12} + P_{13} + P_{14} + P_{23} + P_{24} + P_{34}$$

$$S_3 = P_{123} + P_{124} + P_{134} + P_{234}$$

$$S_4 = P_{1234}$$

$$P_1 = q_{sx} q_{st} \qquad q_{ij} = 1 - p_{ij}$$

$$P_2 = q_{sx} q_{xy} q_{yt}$$

$$\vdots$$

$$P_{12} = q_{sx} q_{xt} q_{xy} q_{yt}$$

$$= P_1 * P_2$$

where $*$ is defined as follows:

$\boxed{q_{ij}^a * q_{kl}^b = q_{ij} \text{ if link } (i,j) \text{ is the same as link } (k,l); a \text{ and } b \text{ are any exponent } \geq 1.}$

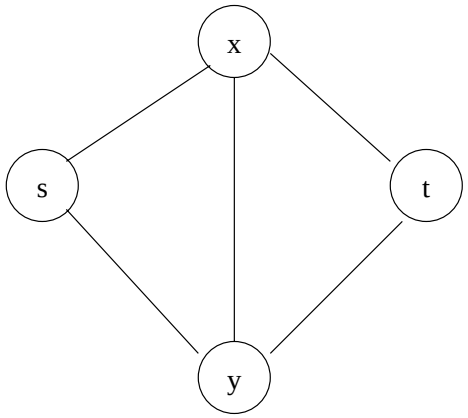


Figure 12.10.

$$P_{13} = P_1 * P_3$$

$$= q_{sx}q_{xt} * q_{sy}q_{yx}q_{xt} \quad \{(x,y) = (y,x)\}$$

$$= q_{sx}q_{xt}q_{sy}q_{yx}$$

$$\vdots$$

Chapter 13. The Arpanet

[As with some of the other chapters the information in this chapter is quite dated. I have left it here for historical interest as much as anything but have not had time to include the figures.]

13.1. Introduction

The network was originally conceived by the Advanced Research Projects Agency (ARPA) or the U.S. Department of Defence as an experiment in computer resource sharing: a network that would interconnect dissimilar computer throughout the United States, allowing users and programs at one computer centre to access reliably and interactively use facilities other centres geographically remote from one another.

The ARPA network has probably generated more interest and excitement in the field of computer networking than any other network in the world. It has spawned a vast amount of research activities in many diverse areas such as: computer-to-computer protocol, interconnection of dissimilar networks, line protocol, communication processor hardware and software design, network topological design, network reliability, network security, adaptive routing and flow control, protocol verifications packet-switching concepts, and so on. The development of the ARPA network has led, directly or indirectly, to the development of a whole host of large-scale computer-communications networks worldwide, both commercial as well as government-owned.

The network began operation in 1969 with four nodes (computer center sites) interconnected; 10 nodes by 1970; 15 nodes by 1971; 24 nodes by January 1972; 34 nodes by September 1972 and 57 nodes by June 1975. Evolution of the ARPANET topology is shown in figure 13.1. The network has been operating ever since, and has subsequently grown to well over 100 computers spanning half the globe, across United States, to Hawaii and to Norway.

Much of our present knowledge about networking is a direct result of the ARPANET project.

13.2. Topology, Concept and Functional Descriptions

The distributed store-and-forward system was chosen as the ARPANET communications system. Such a system has a switch or store-and-forward center at every node in the network. Each node has a few transmission lines to other nodes; messages are therefore routed from node to node until reaching their destination. Each transmission line thereby multiplexes messages from a large number of source-destination pairs of nodes. In addition, emphasis was placed on the segmentation of messages from a computer system (Host) at a source node into blocks called packets and these packets are stored and forwarded to the destination.

The topological connection is in the form of a distributed network which provides protection against total line failure by providing at least two physically separate paths between each pair of nodes. Each HOST is connected through an asynchronous serial 100 kbps channel to a nodal

Figure 13.1.

switching computer called an Interface Message Processor (IMP). The IMP's are themselves interconnected by leased 50 kbps full duplex synchronous channels. In addition, there are two 230.4 kbps lines in California, one 50 kbps satellite channel to Hawaii, one 7.2 kbps satellite channel to Norway, and one 9.6 kbps channel from Norway to England. The IMPs act as network doorways for the HOST computers. The IMP was introduced to relieve the HOST from many of the message-handling tasks of the communication network. In each HOST, a program referred to as the Network Control Program (NCP) is implemented as part of the operating system; the NCP allows the HOST computers to communicate with each other according to a HOST-HOST protocol, which is a network-wide standard. In addition, a program known as TELNET acts as a convenient interface between the user and the NCP, allowing him to converse with the network in a neutral way. Thus the network comprised two parts: a network of data processing systems (HOSTS), and a communication sub-network of packet switching node computers (IMPs).

The ARPANET implementation of a packet-switching network is as follows. For a byte stream from (say) a terminal user to be sent to a remote computing system, the user's HOST must package the byte stream into a message stream. This originating HOST then delivers each message, including a destination HOST address, to its local IMP. The network IMPs then determine the route, provide error control, handle all message buffering and transmission functions, and finally notify the sender of the eventual receipt of the message at its destination HOST.

A dedicated path is not set up between HOST computers that are in communication but rather this communication involves a sequence of message transmissions sharing the communication line with other messages in transit. Thus a pair of HOSTs will typically communicate over the net via a sequence of messages whose maximum size is 8063 bits (plus 40 bits used as a HOST-HOST header). In fact, the IMP, having received the message from the source HOST, partitions the message into one or more packets, each containing at most 1008 bits. Each packet of a message is transmitted independently to the destination IMP which reassembles the message before shipment to that destination HOST. Following this, an end-to-end acknowledgement is sent back to the source HOST. In its journey through the net, a packet will hop from IMP to neighboring IMP; if the neighbor accepts the packet (i.e., the transmission is found to be valid and there is storage available), then an IMP-to-IMP packet acknowledgement is returned. If no acknowledgement is received after a time out period, then the packet is retransmitted. The packet store-and-forward operation and its associated flow control are depicted in figure 13.2.

13.3. Hardware Implementation of Switching Nodes

(1) The original IMP was constructed using a Honeywell DDP-516 minicomputer with l2K 16-bit words of memory. Later Honeywell DDP-316 minicomputers with 16K memories were used.

The IMP is responsible for all the processing of packets, which includes decomposition of HOST messages into packets; routing; relaying and receiving store-and-forward packets; acknowledging accepted packets and retransmitting unacknowledge packets; reassembling

Figure 13.2. Flow control at network level

Figure 13.3. The ARPA IMP

packets into messages at the destination HOST; generating control messages. The IMP is also responsible for gathering statistics, performing on-line testing, and monitoring the line status.

In terms of performance, the 516 IMP can process approximately 850 KBPS and the 316 IMP can process approximately 700 KBPS under the most favourable assumption of maximum length messages The effective capacity of the 50 KBPS line is approximately 40 fullpacket messages per second.

(2) The Terminal IMP (TIP) is built around a Honeywell 316 minicomputer with 28K of memory. It is fitted with a multiline controller (MLC) which allows direct connection of up to 63 terminals to the net, and up to three modem and/or HOST interfaces may be connected to the TIP. Figure 13.4 shows a Terminal IMP connection. The terminals are handled on a per-character basis with start and stop bits even on synchronous lines. Data rates (from 75 BPS up to 19.2 KBPS) and character bit lengths may be set for each terminal by the TIP program. For each input and output terminal line, two full characters are buffered - the one currently being assembled or disassembled and one further character to account for memory accessing delays.

(3) The Satellite IMP, a modified version of the IMP discussed earlier, was developed specifically for satellite links. The Satellite IMP uses some of the packet broadcast ideas that enable several network nodes to statistically share a communications channel to a satellite.

(4) A multiprocessor IMP called Pluribus, based on the Lockheed SUE minicomputer, has been used to provide high reliability and throughput. The Pluribus incorporates 14 processors and can handle tens of hosts and hundreds of terminals simultaneously.

13.4. Host-to-Host or Transport Protocol

The HOST-to-HOST protocol of the ARPANET operates by establishing and clearing connections, giving the network the appearance to the users of a circuit switched system. Since the network provided highly reliable and sequenced message delivery, the host-host protocol focussed on flow control, interrupt, multiplexing, and convenient addressing. The first transport protocol of the ARPANET is the Network Control Protocol (NCP) whose basic functions include open, close, send, receive, and interrupt operations. The user interface to the NCP is stream-oriented, with flow control in both bits and messages provided by incremental “allocate” control messages between NCPs. Interrupt signals are not synchronized with the data stream. Control messages for all connections between a pair of NCPs are sent over a single control connection, while data messages are sent on their individual connections.

Figure 13.4. Terminal IMP connections

Before processes can send messages connections must be established. The basic transport service provided by NCP to its users is an error-free, sequenced, simplex connection. To achieve full-duplex communication, two connections are required. A connection is established between a specific socket at one end and a specific socket at the other end, through which data may be sent in one direction only. The sockets are identified by a 32-bit HOST number, with even number used for receiving and odd numbers for sending. When a connection is set up, the receiving HOST allocates a link number which is used together with the HOST number to identify the connection. The link number is used in subsequent flow control commands.

Figure 13.5 shows diagrammatically the way that processes are linked via the HOST-HOST protocol. The correspondence between processes in the HOST and sockets is maintained by the HOST's NCP.

A process, as shown, may have several send or receive "ports" each identified with a particular socket. The connections shown are functionally like circuit connections, but really they take the form of entries in tables, used to handle messages from one process to another.

The NCPs in the HOSTs control the communication between different HOSTs through the medium of the intervening network. Each NCP is able to receive messages from the other NCPs using its own "link Q".

A connection between two processes in different machines is established by arranging for their NCPs to exchange requests for a connection over the link zero channel, which is assumed to be always open and therefore does not need socket numbers. The control commands used to establish and break connections are STR (Sender To Receiver), RTS (Receiver To Sender) and CLS (Close). When a user does an INIT (a transport service primitives), the transport station inspects the genders (send or receive) of the ports to determine whether the connection is for outbound or inbound traffic. Armed with this knowledge it sends an RTS or STR to the remote host. When the request for connection arrives at the remote host, the transport station there checks to see if anyone is listening on the specified port. If sos an STR or an RTS is sent back; if not the transport station may choose to queue the request for a while, or it may reject the connection attempt using CLS message. Once a connection has been established, it is assigned an 8-bit link number to eliminate the need to send tvio 32-bit sockets in every message.

The NCP does not have any acknowledgement scheme at all; it simply assumes that the subnet can handle al1 errors transparently, and that all messages are delivered in sequence, exactly as sent.

Connections-are closed by an exchange of CLS messages.

13.5. Imp-to-Imp or Network Layer Protocol

Each message is segmented by the IMP into at most 8 packets with a maximum of 1008 bits each. In order to offer virtual circuit service, the ARPANET layer 3 protocol contains provision for explicit end-to end acknowledgements within the subnet. When a message enters the source IMP it is assigned a number. If an acknowledge for this message fails to arrive from the destination

Figure 13.5. Connections and sockets in the ARPA HOST-HOST protocol

IMP within the timeout interval (30 sec), the source IMP sends a query to the destination IMP. Depending on the answer, if any, the source DMP may retransmit the message.

Originally, the routing in the ARPANET was done with the distributed adaptive algorithm. Every 640 msec each IMP exchanged delay information with its neighbours. As the 1160- bit routing messages came in (every 640 msec), each IMP updated its routing tables. In addition, a separate check was made to see which IMPs were reachable by using hop acount as metric. It was found (in 1979) that this algorithm adapted too slowly, occasionally caused packets to loop for long periods of time, did not use alternate routes at all, and that the routing packets were interfering with the regular traffic. It is replaced by a new routing algorithm in which each IMP maintains pertinent routing information of the entire ARPANET, including delays on each line.

Using this data base, every IMP computes the shortest path to every other IMP, with delay being the metric for distance. These paths are used for routing. Since every IMP runs the shortest path calculation on almost the same data base, the paths are consistent and there is little looping. The new algorithm adapts to traffic changes by updating information at each IMP every 10 sec. Each IMP measures the delay on each of its lines average over a 10 sec. period. The results of these measurements, together with an update sequence number, are then broadcast to all other IMPs using a flooding algorithm.

The flow control mechanism in the ARPANET was modified to avoid the occurrence of reassembly lockup. In version 2, no multipacket message ils allowed to enter the network until storage for that meSsage has been allocated at the destination IMP. As soon as the source IMP takes in the first packet of a multipacket message, it sends a small control message (REQALL) to the destination IMP requesting that reassembly storage (i.e., 8 buffers) be reserved. It does not take in further packets from the host until it receives an allocation message (ALL) in reply. To provide high bandwidth for long sequences of messages (i.e. file transfer) only the first message in a sequence of multipacket messages is required to go through the reservation procedure; from that point on, the reservation is held for the message sequence as long as more data is delivered to the source IMP within 250 msec after the RFNM is received (Fis.8). To provide good response for single-packet messages, a special mechanism was provided: a single-packet message serves as its own allocation request. If there is buffer space available at the destination, the message is accepted and passed on to the HOST, then the RFNM is sent back to the source IMP. If no space is available at the destination IMP, this single-packet message is then discarded and this transmission is then considered to be a request for space; when space becomes available, the source IMP is so informed, at which point the stored single-packet may be retransmitted.

Format of an ARPANET data packet is shown in figure 13.6. It can be seen that the ARPANET protocols do not make a clear distinction between layers 2 and 3, and that the ARPANET also has the subnet end-to-end protocol. Seq, Logical channel number, Acknowledgement bits fields belong to the data link protocol (layer 2) most of the rest are used by the source IMP to destination IMP (subnet end-to-end) portion of the protocol.

Figure 13.6. Format of an ARPANET data package

13.6. Data Link Protocol

The ARPANET has both single frame and multiframe messages. Long messages are partitioned into segments each of maximum length 1008 bits. The source IMP first builds a 128 bit-(8 words) header and attaches it to the front of the segment, forming a packet. When an IMP wants to send a packet to an adjacent IMP, it sets it up in a memory buffer and then starts the transmission hardware. Before sending the first character in the buffer, the transmission hardware sends a SYN (SYNchronize), a DLE (Data Link Escape) and an STX (Start of TeXt). These three characters serve to define the start of the frame for the receiver. The contents of the IMP buffer is then transmitted, I bit at a time. Finally the hardware automatically transmits a DLE and an ETX (End of TeXt) followed by a 24 bit CRC checksum and then another SYN character. If there is no new frame to transmit, the hardware keeps sending SYNs. The frame format is shown in figure 13.7.

In order to achieve data transparency a technique called "character stuffing" is used (figure 13.8). That is, to differentiate a data pattern which happens to be DLE ETX (not the ending of the frame) the transmitter hardware automatically inserts an extra DLE in front of any DLE pattern in the text. The receiving hardware strips off the extra DLEs.

The protocol does nct use NAKs. If a packet comes in damaged it is just ignored. The sender will time out in 125 msec, and send it again.

Although the IMP transmitting hardware automatically generates checksums that are verified by the receiving IMP hardware, these checksums only guard against transmission errors. To guard against errors caused by bad memory in an IMP, the source IMP computes a software checksum. This checksum is transported along with the packet. At each intermediate IMP along the way, the software checksum is explicitly verified before sending the IMP-IMP acknowledgement. With 16 bits of software checksum and 24 bits of hardware checksum per packet, the mean time between undetected errors should be centuries.

Figure 13.7.

Figure 13.8. Data transparency by character stuffing

Chapter 14. The Internet Protocol (IP)

14.1. Introduction

"IP", the "Internet Protocol", is the network layer protocol associated with the popular "TCP/IP" network software. IP is the basis of the world-wide network commonly known as the Internet. More correctly the Internet is a connection of smaller networks (an internetwork) and IP is the protocol used to route between those networks. In practice, IP is also used within those networks.

14.2. IP Header

Figure 14.1 shows the format of the IP header. The IP header contains the following fields:

version
: This 4-bit field contains the version number of IP to which this packet conforms. This field should currently contain the value 4, although IP version 6 is currently being defined. Parts of the header for version 6 will be different, particularly the length of the address fields.

IHL This 4-bit field contains the length of the header in 32-bit words. If there are no options, the value of this field will be 5 (giving a header length of 20 bytes).

type of service
: This field gives information about the quality of service requested for this packet. It contains subfields which indicate the type of packet and its urgency.

total length
: This 16-bit field gives the total length of this packet in bytes.

identification
: The identification field is used in conjunction with the source and destination address fields to ensure that each packet is uniquely identified. This field can be used to reassemble packets which have been fragmented because they are too long for one of the links.

flags
: This 3-bit field contains three flags, only two of which are currently defined. One flag is used to indicate that this packet cannot be fragmented. This might be used when the destination node is known to be unable to reassemble fragmented packets. The other flag is used to indicate that this is part of a packet which has been fragmented by the system. This bit is clear if this is the last fragment of a larger packet.

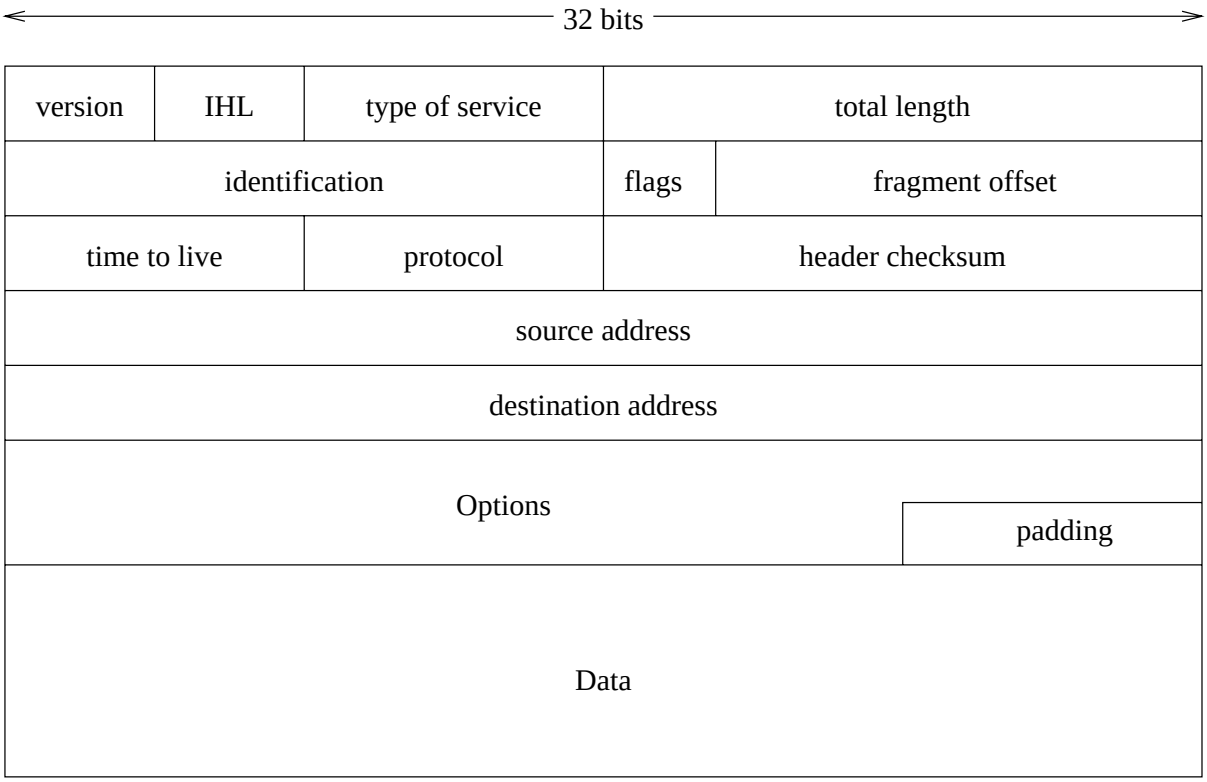


Figure 14.1. The IP packet format

fragment offset
: This field is used when packets are fragmented. It contains the offset from the beginning of the original packet where this packet starts. It is measured in multiples of 8 bytes.

time to live
: This field is initialised when the packet is generated and decremented as the packet passes through each node. If the value ever reaches zero, the packet is discarded. This is intended to defeat routing loops.

protocol
: This field indicates which higher level protocol should be used to interpret the data in this packet. This often corresponds to TCP or UDP, described in the next chapter, but could also be another transport layer protocol.

header checksum
: This checksum is used to ensure that the header has been transmitted correctly.

source address
: This field contains the IP address of the originating host for this packet. This does not necessarily correspond to the address of the node which sent this packet within the network but is the address of the host which first put this packet into the network. It thus differs from the data link layer address.

destination address
: This is the address of the host for which this packet is ultimately destined. It is this address which is used to determine the path this packet will take through the network. Not that each packet contains full addressing information rather than a virtual circuit number. IP is a datagram oriented (connectionless) protocol.

options
: There are various options which are available. We will not worry about them here.

padding
: The padding field is used to ensure that the header is a multiple of 32 bits long.

data
: This field contains whatever data the transport layer passes to the IP layer. The information in this field is not interpreted by the IP layer.

14.3. IP addresses

All IP addresses are (currently) 32 bits. The next version of IP will extend this to much longer addresses. The address consists of two parts. A network number and a host number within that network. To allow maximum flexibility the network and host numbers are of different lengths as described in the subsections which follow.

The format of IP addresses is shown in Figure 14.2. An IP address is traditionally written as four decimal numbers separated by dots with each number representing one byte of the IP address. Thus a typical IP address might be 131.172.44.7.

14.3.1. Class A addresses

A class A address has the most significant bit 0. The next seven bits contain the network number and the last 24 bits the host number. There are thus 126 possible class A networks, each having up to about 16,000,000 hosts. (Networks 0 and 127 are used for other purposes.)

14.3.2. Class B addresses

A class B address has the two most significant bits 10. The next fourteen bits contain the network number and the last 16 bits the host number. There are thus 16384 possible class B networks each containing 65354 hosts.

14.3.3. Class C addresses

A class C address has the three most significant bits 110. The next 21 bits contain the network number and the last eight bits the host number. There are thus more than 2000000 possible class C networks each containing 254 hosts.

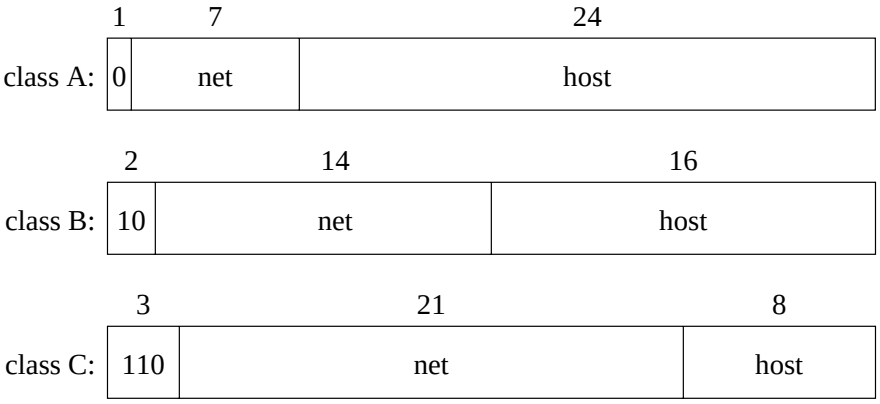


Figure 14.2. IP address formats

14.3.4. Multicast addresses

It is often desirable to send some types of message to many different hosts simultaneously. The remaining IP addresses (i.e. those which start with 111) are used for such messages. As a special case, the address 255.255.255.255 is a broadcast address so that packets with that destination address are received by all hosts within the network. In some networks the address 0.0.0.0 is also a broadcast address.

Within a network, the address with the host number containing all 1s (or 0s) is a broadcast address within that network.

14.3.5. Subnetting

Within an IP network (i.e. that groups of hosts which have the same net number in their IP address), it is still necessary to be able to route between the hosts. For convenience, particularly in class A and class B networks, it is often needed to split the network up into smaller parts and then route between those parts much as the routing is done at a higher level between networks.

Accordingly many IP networks are "subnetted". That is, there is another partition made in the IP number in that the host number is split in two parts, a subnet number and a host number. The partition is made at a convenient point given the topology of the desired network. Effectively, the boundary between the host and net numbers is moved to the right, but only within the network. As far as hosts outside the network are concerned there is no subnetting within other networks.

14.4. Routing IP

Because an internet is a collection of networks, routing is simplified by the assumption that a host in a network can connect to any other host in the same network without leaving that network. Thus if two hosts have IP addresses with the same network number then they can communicate without leaving that network. The further assumption is made that if a host from outside that network knows how to get to any one host in a particular other network then that other host will be able to route packets to all hosts in that network. This drastically simplifies the routing problem because now all routers in the network only need to know how to get to a particular network and not to every host within each network.

Unfortunately, because the Internet is so successful, there are now a very large number of networks connected together. Because there is no extra routing information in the net number part of the IP address, all routers need to remember how to get to every one of the networks constituting the internet. Routers thus need large amounts of memory to contain these tables and a significant amount of time is spent in transmitting routing information between routers. Some changes have been made in the way in which IP numbers have been allocated in an attempt to partly alleviate this problem and IP version 6 will be arranged so that there is routing information inherent in an IPv6 address.

14.5. ARP

A major problem in networks is finding the address of the host that you want. For example, once you know the IP address of the destination machine you need to work out where to send th packet. Part of this is solved by the routing algorithms described elsewhere, but once the destination machine is known to be on your network (say a broadcast network like ethernet) you still need to find the data link layer address of the desired machine without having to maintain large translation tables.

This problem is solved by using the Address Resolution Protocol (ARP). An ARP packet is shown in figure 14.3. The particular case shown here is for an IP address to ethernet address translation, but the protocol is designed to be able to be used for any address translation required between any two protocol addresses. The packet shown here is a full ethernet packet, including the ethernet addresses and packet type field at the beginning. (The padding necessary at the end to achieve a minimum length ethernet packet is not shown). Note that this is not an IP packet. The ARP protocol is a separate protocol which does not need any routing.

When a host needs to translate an IP address to an ethernet address it constructs a packet as shown in figure 14.3. Obviously it cannot fill in all the fields or it would not need to send the request. Instead, the destination ethernet address field (the first field) contains the ethernet broadcast address (all ones) and the target hardware address field (which contains the value we are trying to find) is left blank.

This packet is transmitted and any host on the network which knows the desired ethernet address (most probably the target host) will fill in that field, change the opcode field to indicate that this is a reply, modify the destination and source ethernet addresses at the beginning and transmit the packet. The original host can then read the reply to find the desired ethernet address.

14.6. RARP

A different, but related, problem to that of finding another machines ethernet address when you know its IP address is to find you're own IP address when all you know is your ethernet address. The ethernet address of a host is often configured into its hardware somewhere, but its IP address will be assigned by an administrator somewhere within the network. It is inconvenient to have to set up each individual computer to tell it its IP address and this procedure is also prone to error.

An alternative is to maintain a central database containing the IP addresses of all hosts in a network and let them ask for their addresses. (This also allows for dynamic allocation of IP addresses when only a few hosts are likely to be using the network at the same time.) This problem is compounded by the hosts inability to send packets to other machines because it doesn't know

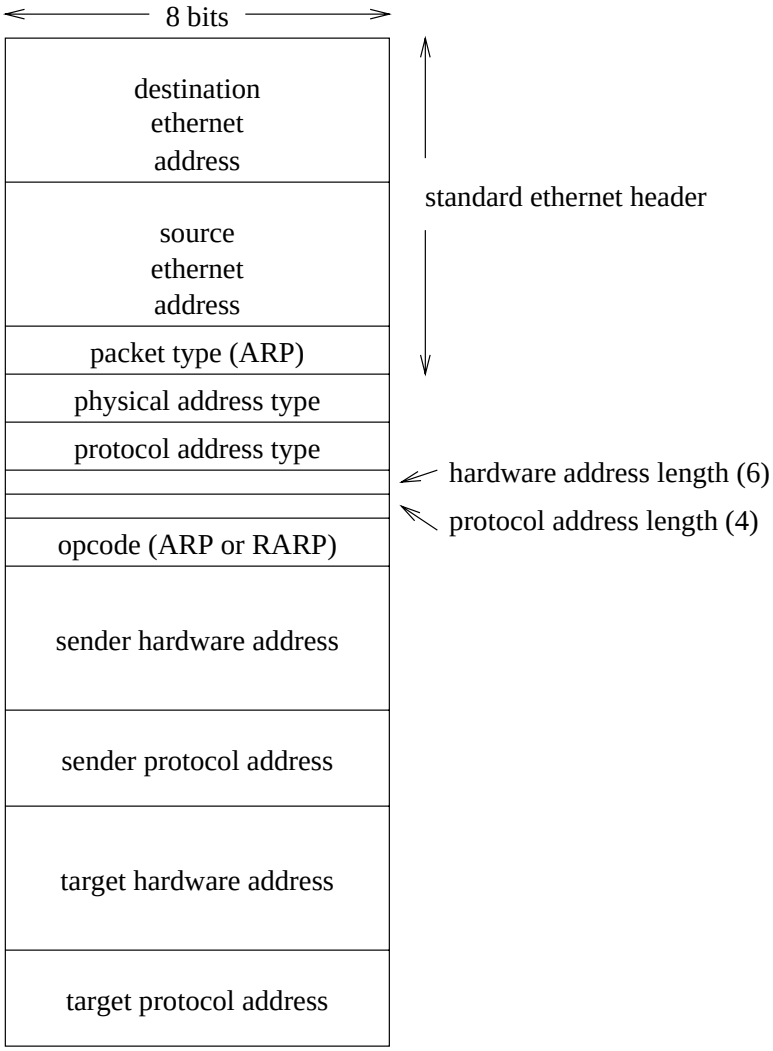


Figure 14.3. ARP/RARP packet format

what to put in the source address fields.

This problem is solved by modifying the ARP protocol slightly. This alternative is the Reverse Address Resolution Protocol (RARP). A packet similar to the ARP packet is constructed, the only differences being that the sender protocol address field is left blank (as well as the target addresses) and the opcode field is changed to represent a RARP request. The packet is broadcast onto the network and the database server fills in the blank fields and sends the response.

14.7. ICMP

Within an IP internetwork there is often the need for auxiliary information to be transmitted between hosts relating to the operation of the network rather than communication between the hosts. The Internet Communication Message Protocol (ICMP) is designed for this job and it has messages which can send information through the network passing information such as the reasons why packets are dropped.

It also contains an ECHO directive which simply asks the target machine to send it back. This

is useful to see whether a remote host is still working.

Chapter 15. TCP and UDP

15.1. Introduction

There are several transport layer protocols used with IP. The most common of these are TCP (thus giving rise to the common term "TCP/IP") and UDP.

15.2. TCP

The Transmission Control Protocol (TCP) is one of the main transport layer protocols used with IP. It is a connection oriented protocol based on the connectionless IP protocol. Because it is the lowest layer which has end-to-end communication, it needs to handle things such as lost packets. In this respect it is similar to the data-link layer which must handle errors on an individual link. Consequently many of its facilities are familiar from our discussion of the data-link layer.

The format of a TCP header is shown in figure 15.1.

source port
: All of the address fields in the lower layer protocols are only concerned with getting the packet to the correct host. Often, however, we want to have multiple connections between two hosts. The source port is simply the number of the outgoing connection from the source host.

destination port
: Similarly, this is the number of the incoming connection on the destination host. There must be a program on the destination host which has somehow told the networking system on that host that it will accept packets destined for this port number. Standard system services such as SMTP, NNTP and NTP (all described in later chapters) have well known standard port numbers. Thus to connect to the SMTP port on a particular host (in order to transmit some email) a TCP connection would be set up with the correct destination port number (25). The source port number does not matter except that it should be unique on the sending machine so that replies cvan be received correctly.

sequence number
: This is the sequence number of this packet. It differs from the usual data-link layer sequence number in that it is in fact the sequence number of the first byte of information and is incremented by the number of bytes in this packet for the next message. In other words, it counts the number of bytes transmitted rather than the number of packets.

acknowledgement number
: This is the sequence number of the last byte being acknowledged. This is a piggy-backed acknowledgement.

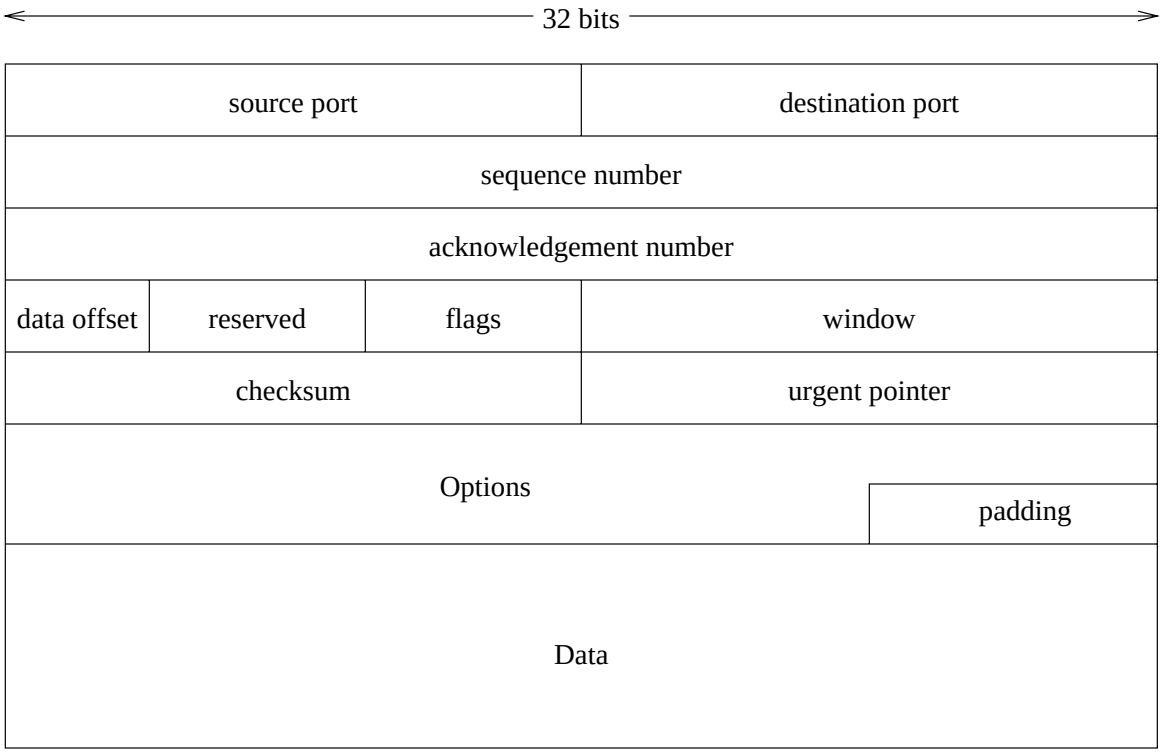


Figure 15.1. TCP header format

data offset
: This field is the offset in the packet of the beginning of the data field. (In other words it is the length of the header.)

flags
: This field contains several flags relating to the transfer of the packet. We will not consider them further here.

window
: This field is used in conjunction with the acknowledgement number field. TCP uses a sliding window protocol with a variable window size (often depending on the amount of buffer space available. This field contains the number of bytes which the host is willing to accept from the remote host.

checksum
: This field contains a checksum of the header. It actually uses a modified form of the header which includes some of the information from the IP header to detect some unusual types of errors.

urgent pointer
: There is provision in TCP for some urgent data messages to be sent bypassing the normal sequence number system. This field is used to indicate where such data is stored in the packet.

options
: As with IP, various options may be set. We will not consider them here.

padding
: Padding is added to make the header a multiple of 32 bits long. This is only necessary when options are used.

data
: The data field is passed intact to the program which is receiving packets addressed to this port.

15.3. UDP

The User Datagram Protocol (UDP) is a datagram transport which uses the underlying IP protocol for its network layer. It is used when there is a need to transmit short packets through a network where there is no stream of data to be sent as in TCP. It is consequently a much simpler protocol and therefore much easier to handle. It is also less reliable in that there are no sequence numbers and other error recovery techniques available. If errors and lost packets are important then the user of UDP must cater for these.

The format of a UDP header is shown in figure 15.2.

source port
: This field performs the same function as in TCP.

destination port
: This field is also the same as in TCP. Note that the same port number can be used by TCP and UDP for different purposes.

length
: This field contains the length of the data field.

checksum
: As in TCP, this field is the checksum of the header plus parts of the IP header.

data
: This field is passed to the relevant program.

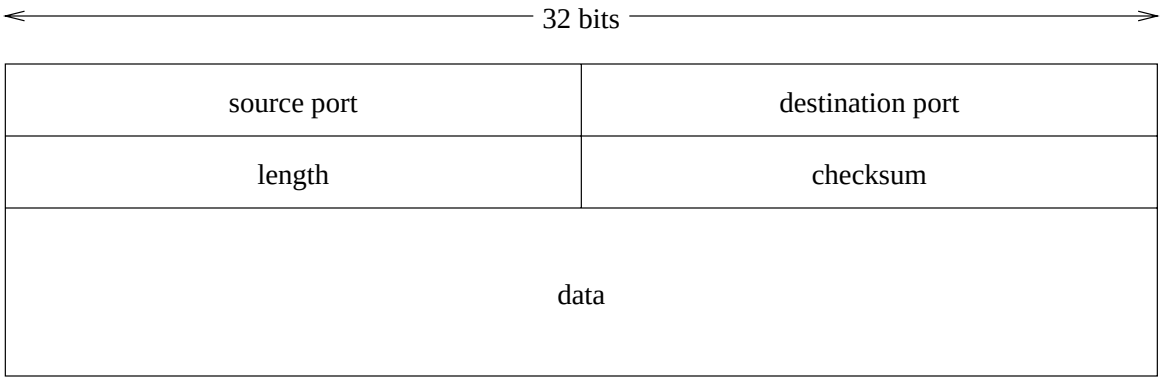


Figure 15.2. UDP header format

Chapter 16. Who's Who (the Domain Name System)

16.1. Introduction

While IP addresses uniquely designate a host connected to the Internet, they are not convenient to use because they are difficult to remember, containing no mnemonic information. As people we would rather remember names which have som meaning to us. Accordingly most computers have names. The main School of Electronic Engineering Unix server, for example, is called "faraday". Unfortunately, there are so many computers in the world that it is difficult to come up with a unique name for each computer, and so computers are grouped together into domains. In fact a hierarchical structure is employed. The full name of each computer (its fully qualified domain name or FQDN) is formed by joining the name of the computer together with its doamin name. Thus faraday's full name is faraday.ee.latrobe.edu.au indicating that it is in the School of Electronic Engineering (ee) at La Trobe University (latrobe) which is an educational institution (edu) in Australia (au).

Faraday also has an IP address (some computers and all routers have more than one) of 131.172.44.7. It is the job of the doamin name system to translate between the two. Note that there is complete orthogonality (at least in principle) between the domain name system and IP addresses. While shannon is another computer in the School of Electronic Engineering it could have a quite unrelated IP number. In practice, the two systems tend to be fairly well correlated simply because of the way in which the two sets of naming procedures are set up. (Shannon's IP address is 131.172.44.4.)

16.2. Host tables

Once upon a time (i.e. only a few years ago) all the hosts connected to the Internet were listed in a file which contained both there FQDNs and their IP addresses. Every now and again (i.e. once a week or so) every computer connected to the network would obtain a new copy of the file. Whenever a translation needed to be made it was a relatively simple job to look up the relevant data in the file.

As the Internet grew, this quickly became unwieldy. The file grew so large and changed so rapidly that it became impossible to keep it up to date and the time needed to transfer it around the world increased so that it would take up a significant part of the network bandwidth. The administration required to update also became very difficult.

Accordingly this system is no longer used, being replaced by the domain name system.

16.3. Named

The domain name system is really a world-wide distributed database. (In some respects it is amazing that it works at all.) The records for each subdomain are kept by the administrators of that domain and when a host needs to translate a FQDN to an IP address it sends a request to the host which contains that part of the database to find that information.

That is a simplistic view of the system. It is complicated by having to discover just which host it is that holds the right part of the database. In practice, the system works as follows:

When a translation is required (say to translate faraday.ee.latrobe.edu.au), the program named on the host will connect to one of a few well known hosts which know all about the top-level domains, asking for information about the ee.latrobe.edu.au domain. The answer will probably be that it knows nothing about that doamin but it does know who to ask about the au top-level domain. That machine will be contacted to find out about the edu.au domain which will give another pointer to the latrobe.edu.au domain and so on until it finds a server which knows the information we are seeking.

While this is a fairly long procedure, the information found will be cached so that if a similar search is undertaken it will proceed much more quickly.

A reverse search is made possible by the addition of another top-level domain. If it is desired to find the FQDN corresponding to faraday's IP address, then a search would be made for "7.44.172.131.in-addr.arpa". The same procedure described above can then work through the hierarchical system to find the corresponding name.

While this sounds fairly simple, the actual protocols are more complicated to enable the system to still work when parts of the network are not working correctly and to enable the system to work with as little network load as possible. In addition, the DNS is also able to distribute information other than just FQDN to IP adress translations. For example, it also contains so-called MX records which contain information about where to send email. For example, an email address like K.Pye@ee.latrobe.edu.au will work even though there is no computer called just "ee.latrobe.edu.au". There is however an MX record which says that email should be sent to faraday, which understands that it should deliver that email as though it were addressed to "kjp@ee.latrobe.edu.au".

Chapter 17. SMTP and Mail

17.1. Introduction

The Simple Mail Transfer Protocol (SMTP) is the protocol used by TCP/IP networks for transferring email between systems. It is a mail transfer agent (MTA) as distinct from a mail user agent (MUA) which is used by real people to read and send mail. A MUA will use a MTA to actually transfer the mail to the correct host on which the intended recipient will use a (possibly different) MUA to read and respond to the mail.

Email is a store-and-forward system. The email is not necessarily delivered immediately to its recipient or even to the final machine. It is instead sent at least part of the way to its destination, where it is stored until it can be forwarded further on its journey. This is done for mainly historical reasons when telecommunications were expensive and not all hosts were connected together all the time, but instead dialled each other at regular times and transferred any messages which were waiting for them. Today, most systems are permanently connected and so email travels much more quickly. Indeed automatic response systems can sometimes have replies back for mail to them within seconds. In any case, email must be a form of store-and-forward system since we cannot guarantee the intended recipient is waiting patiently at their computer for your email to arrive, and so the email must always be stored at its destination.

17.2. Mail format

Real ("snail") mail contains four basic parts: the envelope, the headers, the body and the signature. There can also be attachments to it.

Email is modelled after ordinary mail, and also has all the concepts of ordinary mail.

The *envelope* contains the mail, and contains all the information needed to transmit the message to its destination. It may be modified during transmission by the addition of such information as postmarks to indicate where the mail has been and how it reached its destination. It might also contain redirection messages.

The *headers* are contained as part of the actual message, and contain information such as the identity of the sender and recipient (including their addresses) and the date.

The *body* of the message contains the usual information. Nothing in the body is important for the transmission of the message.

The *signature* repeats some of the information in the header about the sender, but more importantly it can be used to authenticate the sender (i.e. to prove that the sender really is who they claim to be).

Attachments may be included which relate to the letter but are not actually part of it.

Email contains only three major parts, and moves the boundary between some of them. An email envelope contains the information as to the sender and recipient of the mail. The email

headers contain all of the information of the ordinary mail headers but add to that the information sometimes added to the envelope of ordinary mail relating to the transmission of the mail. Email also contains no specific signature and attachments but there are methods of including this information in the body of a message. An email message thus consists of only three parts, the envelope, headers and body.

The headers and body are normally stored in a single file. The header is all the lines at the beginning of the file up to the first blank line. Everything else is the body. The header lines are all of the same format. They consist of a keyword, followed by a colon and space followed by any extra information needed.

The envelope information is not stored in the file but is passed through the SMTP protocol separately as shown in section 17.4. When the email is delivered at its final destination, the sender information from the envelope is added to the beginning of the file and appears as another header line (although the colon is missing).

17.3. Headers

There are many possible header lines used in mail. Only a few of the most important are described below.

From:
: This header contains the name of the person sending the message. It often contains both their email address and the full name if that information is available to the MUA originating the message.

Sender:
: This header contains the name of the person who actually caused the email to be sent. This might be the secretary who sent the mail after asked to do so by his boss, whose name would appear in the From: header.

To:
: This header contains the name of the person for whom the message is primarily intended. This is not necessarily the same as the person for whom this particular copy of the message is intended. That information is contained in the envelope and is often different when several copies of the message are being sent, as with mailing lists or when the CC: header (see below) is used.

Received:
: This header line is added by each MTA which sees the message as it is passed to its destination. By looking at the Received: headers it is possible to tell how the message travelled through the network and how long it spent on each host (assuming that their clocks are accurate).

Date:
: This header contains the date and time when the MUA first sent the message.

Subject:
: This header contains information which the sender of the message has specified to give the recipient of the message some indication of what the message is about. MUAs would typically display this information as well as the name of the sender so that the recipient can choose the order in which he reads his email (if at all).

Return-path:
: This header should contain a valid email address which could be used to send an answer back to whoever sent this email.

Reply-to:
: This header contains the address of a person to whom replies should be sent. This is not necessarily the same person who sent the mail.

CC:
: This header contains the email addresses of other people who will receive this email as well as the primary recipient.

Content-length:
: This header gives the total length of the message. It is used to distinguish between messages when multiple messages are stored in the same file.

Message-ID:
: This header contains an identification string for this message. It is unique, so that it is possible to tell whether you have seen this message before.

Mime-content-type:
: This header is used to specify the format and encoding of the body of the message. The Multipurpose Internet Mail Extensions (MIME) are a set of standard ways of organising the body of a mail message to handle signatures and attachments as well as transferring non-textual information within a mail message.

17.4. The protocol

In this section we describe only a small part of the SMTP protocol. We will do so by following through a simple SMTP session, sending a simple mail message. This is an unrealistic example in that the sender and recipient addresses are on the same machine. We assume that a TCP connection has been made to the SMTP port on faraday.ee.latrobe.edu.au. Each of the commands given here will generate a response from the destination host. We assume here that each response is favourable and that we should continue.

HELO faraday.ee.latrobe.edu.au
: This command simply introduces the source host to the destination host. The destination host might verify that this is a valid machine name and that it does in fact correspond to the host who is connected. This is done to make forgeries more difficult.

RCPT TO: K.Pye@ee.latrobe.edu.au

This command says that the host wants to transmit a message to the specified email address. That address need not be on the destination machine, which would then forward it on.

MAIL FROM: kjp@ee.latrobe.edu.au

This command gives the email address of the originator of the message. Note that this command and the previous one transfer the envelope of the message.

DATA

This command is followed by the actual message itself, i.e. the headers and body. They are simply sent as is and followed by a line containing only a single dot which indicates the end of the message. (If the message contains such a line then another dot is added. The extra dot is stripped by the receiving MTA. This is similar to the bit stuffing used by SDLC.)

QUIT

This indicates the we have no more messages to transmit and are about to disconnect.

Chapter 18. NNTP and News

18.1. Introduction

The Network News Transport Protocol (NNTP) has two quite distinct although related functions. One is to allow the transfer of news articles between machines and the other is to allow newsreaders to access those articles from a database maintained on a news server. The commands needed to do these two jobs are quite different and it would probably have been better to design two different protocols, however NNTP is designed to do both.

18.2. Article format

News articles are very similar in format to mail messages as described in the last chapter.

A news article consists of a header, similar in format to a mail header, including the format of each header line, and a body. MIME can also be used to enhance the capabilities of the body of the article. Most of the header lines described in the last chapter can also be used in news articles, although a few, such as the "To:" and "Received:" headers are either meaningless or not used in news articles. In addition the following header lines may be useful:

Path:
: This header replaces the Received: header of email. Each time the article passes through a machine, the name of the machine is placed at the beginning of a list. It is thus possible to trace the path of the article.

Newsgroups:
: This header contains the list of newsgroups to which this article should be posted.

References:
: When a followup is made to a article, the references line of the old article is copied and its message-id added to the end. It is thus possible to trace back and find the thread of articles in which this article belongs.

Lines:
: This is simply the number of lines in the message.

Summary:
: This (often multi-line) header contains a brief summary of the contents of the article. It gives more detail than the subject line.

Distribution:
: This indicates the area in which the article should be distributed. For example, if the article is intended only for an Australian audience it might contain "Distribution: aus".

Followup-To:
: This header indicates where followups should be directed. It might contain a subset of the newsgroups to which the article is posted or it might contain the word “poster” which indicates that responses should be sent back to the sender by email.

Approved:
: Some newsgroups are moderated which means that only one person (the “moderator”) can post to them. This header line indicates that the article has been approved by the moderator.

Expires:
: This indicates when the poster believes that the article should be deleted. It is sometimes used when the information in the article is only relevant for a short time so that the article can be deleted when the information is no longer useful and sometimes used when the information will be useful for a long period of time and should be kept longer than usual. Other articles are deleted when the disk space is needed for newer articles.

18.3. The reading protocol

Because there are a large number of different possibilities for reading news, there are a lot of commands associated with it. Only the most important will be described here.

The reading process is based on newsgroups, just as it is in the newsreader program (similar to a MUA) used by the person reading the news. This simplifies the interface and allows the server to be implemented more efficiently. The command “GROUP news.group.name” is used to change to a newsgroup. The response to this command indicates the range of valid article numbers.

An article can then be retrieved with the “ARTICLE number” command. The requested article (if it is available) is then transmitted, followed by a line containing only a dot, as with the transfer of mail messages using SMTP.

Alternatively, the “HEAD number” command can be used, which will only transfer the headers for the article. This could allow the reader to decide whether he really wants to read this article.

This would normally be followed by the “BODY” command which transfers the rest of the article.

Other commands can be used to transfer the list of all newsgroups with the current article numbers, to transfer information from a database containing some of the header information for all articles and to post article.

18.4. The transfer protocol

The protocol used for transferring articles between servers is much simpler, since the variants are simpler. When a server has news articles to transfer, it will connect to the NNTP port on the other server and send an “IHAVE <message-id>” command. The other server will check to see whether it already has that article (which it might have received from another server) and respond appropriately. If the other server does not have the article it will be transmitted. If more articles

are to be transmitted then the procedure is repeated. When no more articles remain, the connection is terminated with the "QUIT" command.

This protocol is a simple stop-and-wait protocol and suffers all the problems associated with such a protocol. With the increasing volume of news traffic, a new protocol became necessary. This is a pipelined protocol which separates the tasks of seeing whether the other server already has a particular article and actually sending the article.

The first task is undertaken with the "CHECK <message-id>" command. The response indicates whether the server already has that article or not. Several such commands can be outstanding at a time, so that the latency of the network (and the lookup time on the server) does not slow down the overall process.

A separate "TAKETHIS <message-id>" command is used to transfer the articles which the other server does not have. There is no need to wait after the "TAKETHIS" command before sending the data since we already know that the other server does not have the article. (In the rare case where another server has transferred the article in the meantime then one of the copies will simply be discarded.)

Chapter 19. The Network Time Protocol

19.1. Introduction

Often in networks of computers it is important (or sometimes just convenient) for them all to have the same idea of the time. When times are stored for the modification of files it can be important in what order the files were modified. Similarly some protocols (particularly those associated with security) can be dependent on having the correct time (or at least the same idea of the current time). When computers are connected together it is possible for them to exchange their idea of the time and thus to synchronise their clocks.

19.2. The Network Time Protocol

The Network Time Protocol (NTP) is a protocol designed to facilitate the exchange the current time and provide a means for accurately establishing the time over a network of computers.

The time is transferred as a 64 bit fixed-point quantity, with the top 32 bits representing the time in seconds since the beginning of 1900 and the remaining 32 buts representing the fractions of a second, giving a resolution of about 232 picoseconds and a range of 136 years.

UDP packets are exchanged regularly between hosts which contain that hosts idea of the current time together with its estimatiom of how accurate that value is. Some hosts also have external accurate clocks connected to them derived from such sources as atomic clocks and GPS receivers. They advertise an accurate time which the others synchronise to.

To improve the accuracy, statistics are kept about the round-trip delay times between hosts and how much they vary. Various statistical calculations are made which allow an accurate idea of the time to be derived and maintained.

Chapter 20. Security

20.1. Introduction

Whenever a computer is connected to a network, questions of security arise. Any computer is subject to security problems. Who should be allowed to use it? What data should they be allowed to access? How do you prevent data from being corrupted or transferred to somebody who shouldn't have it?

Connecting the computer to the network compounds these problems. Now it is possible for people to access the computer without being physically present. In addition, now some of the data is being transferred over the network and passing through other people's hosts and cables where we have no control. How do we know that the data is coming from where we think it is?

20.2. Authentication

Authentication is the process of verifying that somebody is who they say they are. There are various techniques for doing this relying either on the possession of some unique physical device (including various characteristics of a person such as a fingerprint or retina pattern or a physical device such as an access card) or on the common knowledge of some piece of information which only the person and the computer possess (such as a password).

Unfortunately, when the computers are connected to a network, the authentication information must pass over the network somehow, and it is then prone to capture by other people who then have the information. Even if it is a physical device which is needed, the characteristics of it must pass over the network and if those characteristics are later retransmitted then it can be difficult to tell the difference between the original transmission of the data and a later forgery.

The most common form of authentication today is still the simple password. Passwords are becoming less secure, partly because computers are becoming faster and faster and so it is easier to crack them and partly because they are prone to "sniffing" by observing packets on a local area network. Consequently it is necessary to find replacement techniques for user authentication.

Most of the possible replacements rely on the use of one-time-passwords (OTPs) which are like ordinary passwords except they are valid for only one use.

Some OTP systems rely on a user having a physical device which displays a password which changes regularly. The computer knows which password will be displayed at any time and thus knows which to expect. There are problems with such a system in keeping the time on the portable device and the computer synchronised but they can be overcome by having the computer accept any one of a few passwords generated around the correct time and thus being able to calculate the drift of the clock on the portable device. Such a system does rely on the user not losing the device. If it is lost then the security of the system is compromised because the finder can use it to log in to the computer.

Another OTP system uses encryption to generate the passwords. By using a pass phrase which is remembered by the user and typed in to the computer at which he is actually sitting (and thus not passing over the network) it is possible to generate passwords by repeatedly encrypting the phrase. The first time it is used, the pass phrase might be encrypted 100 times. The next time it would be encrypted 99 times and so on. With a good encryption scheme, knowing the passwords which have already been used it will still be impossible to generate the next password to be used. When the pass phrase has been used 100 times (the last with only a single encryption) it will be necessary to choose another pass phrase. This would need to be done while physically located at the computer (so that the pass phrase does not pass over the network) or encrypted in some way using keys which have already been agreed upon.

A related problem is that of authenticating messages on a network. This particularly applies to email messages where you want to be sure that they really come from whom they purport to. There are now systems available which allow this authentication to take place, although they all share the problem that at some stage you have to trust some information that you receive from somebody else.

20.3. Data Integrity

The integrity of your data is also extremely important. You need to be sure that the data has not been changed, that it is correct (and is reliable) and that nobody sees it who shouldn't. The last of these requires some form of encryption if the data is to pass over a network and the encryption can also be used to help with the other requirements.

The main problem with encryption is passing the keys to those people who need them and making sure that those who shouldn't have them don't. This is similar to the password problem and to the problem of keeping the data secret in the first place. The safest way to transfer the keys is to not use the network, but pass them in some other way.

An alternative is to use a public key system. This is a system where the key used to encrypt the data is different from the one used to decrypt the data and the decryption key cannot easily be derived from the encryption key. Then if I want someone to send me some data I can send them the encryption key (or publish it in some public place) and they can then encrypt the data and send it to me. Nobody apart from me can decrypt the data (even the person who encrypted it). With such a system it also becomes possible to solve the authentication problem with the data. If you send with the request some data encrypted with the other person's public key and they decrypt it and send it back encrypted with your public key then you can be sure that the person whose public key you used is the person who is sending you the data in the same message.

In practice public key systems need to use very long keys to be secure. Keys of thousands of bits are common. This makes the encryption and decryption processes slow. Thus if a large amount of data needs to be encrypted you would probably use the public key system to exchange keys for a simpler and much faster encryption system and use that to actually encrypt the data.

Appendix A. A TCP/IP Tutorial

Status of this Memo[1]

This RFC is a tutorial on the TCP/IP protocol suite, focusing particularly on the steps in forwarding an IP datagram from source host to destination host through a router. It does not specify an Internet standard. Distribution of this memo is unlimited.

A.1. Introduction

This tutorial contains only one view of the salient points of TCP/IP, and therefore it is the "bare bones" of TCP/IP technology. It omits the history of development and funding, the business case for its use, and its future as compared to ISO OSI. Indeed, a great deal of technical information is also omitted. What remains is a minimum of information that must be understood by the professional working in a TCP/IP environment. These professionals include the systems administrator, the systems programmer, and the network manager.

This tutorial uses examples from the UNIX TCP/IP environment, however the main points apply across all implementations of TCP/IP.

Note that the purpose of this memo is explanation, not definition. If any question arises about the correct specification of a protocol, please refer to the actual standards defining RFC.

The next section is an overview of TCP/IP, followed by detailed descriptions of individual components.

A.2. TCP/IP Overview

The generic term "TCP/IP" usually means anything and everything related to the specific protocols of TCP and IP. It can include other protocols, applications, and even the network medium. A sample of these protocols are: UDP, ARP, and ICMP. A sample of these applications are: TELNET, FTP, and rcp. A more accurate term is "internet technology". A network that uses internet technology is called an "internet".

A.2.1. Basic Structure

To understand this technology you must first understand the logical structure shown in figure A.1.

This is the logical structure of the layered protocols inside a computer on an internet. Each computer that can communicate using internet technology has such a logical structure. It is

[1]This appendix contains a copy of RFC1180, written by T. Socolofsky and C. Kale of Spider Systems Limited in January 1991. The only changes I have made here are to reformat the text and redraw the diagrams and minimal changes necessary because of the reformatting.

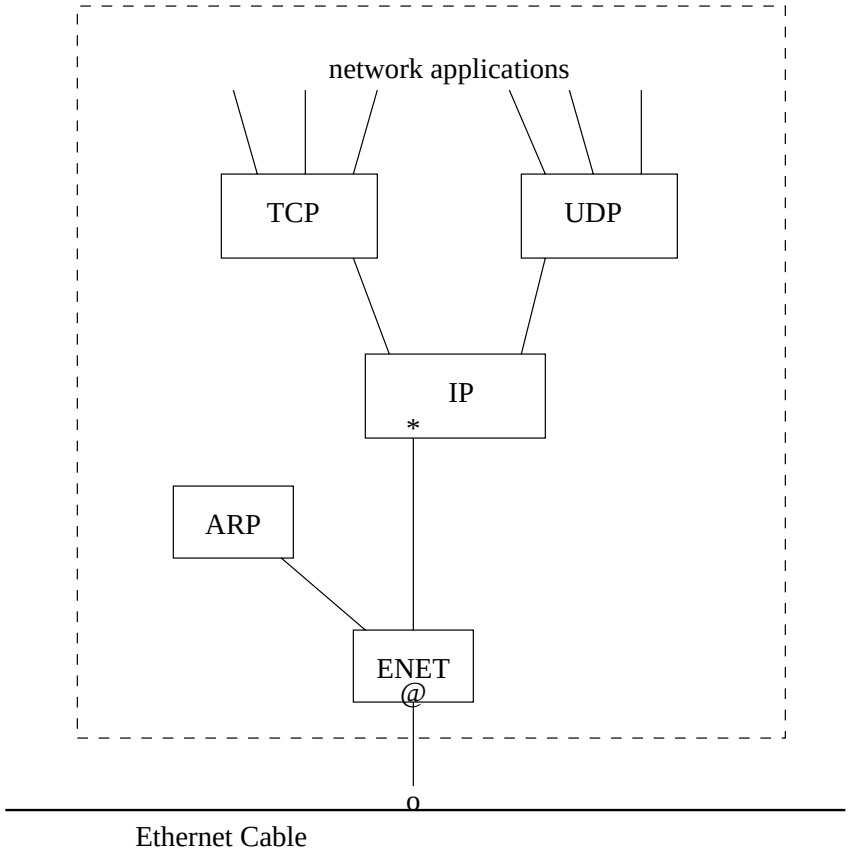


Figure A.1. Basic TCP/IP Network Node

this logical structure that determines the behavior of the computer on the internet. The boxes represent processing of the data as it passes through the computer, and the lines connecting boxes show the path of data. The horizontal line at the bottom represents the Ethernet cable; the "o" is the transceiver. The "*" is the IP address and the "@" is the Ethernet address. Understanding this logical structure is essential to understanding internet technology; it is referred to throughout this tutorial.

A.2.2. Terminology

The name of a unit of data that flows through an internet is dependent upon where it exists in the protocol stack. In summary: if it is on an Ethernet it is called an Ethernet frame; if it is between the Ethernet driver and the IP module it is called a IP packet; if it is between the IP module and the UDP module it is called a UDP datagram; if it is between the IP module and the TCP module it is called a TCP segment (more generally, a transport message); and if it is in a network application it is called a application message.

These definitions are imperfect. Actual definitions vary from one publication to the next. More specific definitions can be found in RFC 1122, section 1.3.3.

A driver is software that communicates directly with the network interface hardware. A module is software that communicates with a driver, with network applications, or with another module.

The terms driver, module, Ethernet frame, IP packet, UDP datagram, TCP message, and application message are used where appropriate throughout this tutorial.

A.2.3. Flow of Data

Let's follow the data as it flows down through the protocol stack shown in Figure A.1. For an application that uses TCP (Transmission Control Protocol), data passes between the application and the TCP module. For applications that use UDP (User Datagram Protocol), data passes between the application and the UDP module. FTP (File Transfer Protocol) is a typical application that uses TCP. Its protocol stack in this example is FTP/TCP/IP/ENET. SNMP (Simple Network Management Protocol) is an application that uses UDP. Its protocol stack in this example is SNMP/UDP/IP/ENET.

The TCP module, UDP module, and the Ethernet driver are n-to-1 multiplexers. As multiplexers they switch many inputs to one output. They are also 1-to-n de-multiplexers. As de-multiplexers they switch one input to many outputs according to the type field in the protocol header.

If an Ethernet frame comes up into the Ethernet driver off the network, the packet can be passed upwards to either the ARP (Address Resolution Protocol) module or to the IP (Internet Protocol) module. The value of the type field in the Ethernet frame determines whether the Ethernet frame is passed to the ARP or the IP module.

If an IP packet comes up into IP, the unit of data is passed upwards to either TCP or UDP, as determined by the value of the protocol field in the IP header.

If the UDP datagram comes up into UDP, the application message is passed upwards to the network application based on the value of the port field in the UDP header. If the TCP message comes up into TCP, the application message is passed upwards to the network application based on the value of the port field in the TCP header.

The downwards multiplexing is simple to perform because from each starting point there is only the one downward path; each protocol module adds its header information so the packet can be demultiplexed at the destination computer.

Data passing out from the applications through either TCP or UDP converges on the IP module and is sent downwards through the lower network interface driver.

Although internet technology supports many different network media, Ethernet is used for all examples in this tutorial because it is the most common physical network used under IP. The computer in Figure A.1 has a single Ethernet connection. The 6-byte Ethernet address is unique for each interface on an Ethernet and is located at the lower interface of the Ethernet driver.

The computer also has a 4-byte IP address. This address is located at the lower interface to the IP module. The IP address must be unique for an internet.

A running computer always knows its own IP address and Ethernet address.

A.2.4. Two Network Interfaces

If a computer is connected to 2 separate Ethernets it is as in Figure A.3.

Please note that this computer has 2 Ethernet addresses and 2 IP addresses.

It is seen from this structure that for computers with more than one physical network interface,

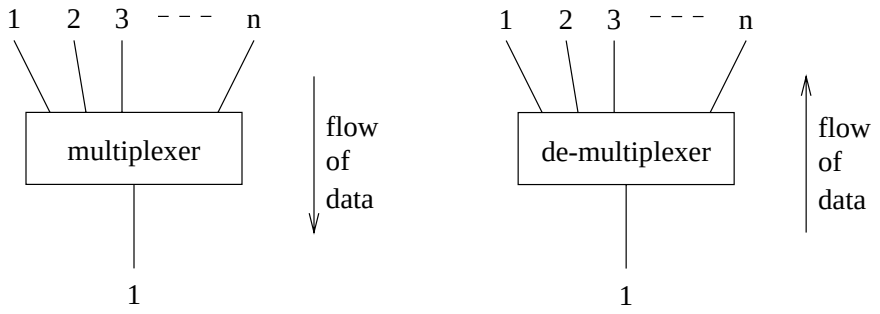


Figure A.2. n-to-1 multiplexer and 1-to-n de-multiplexer

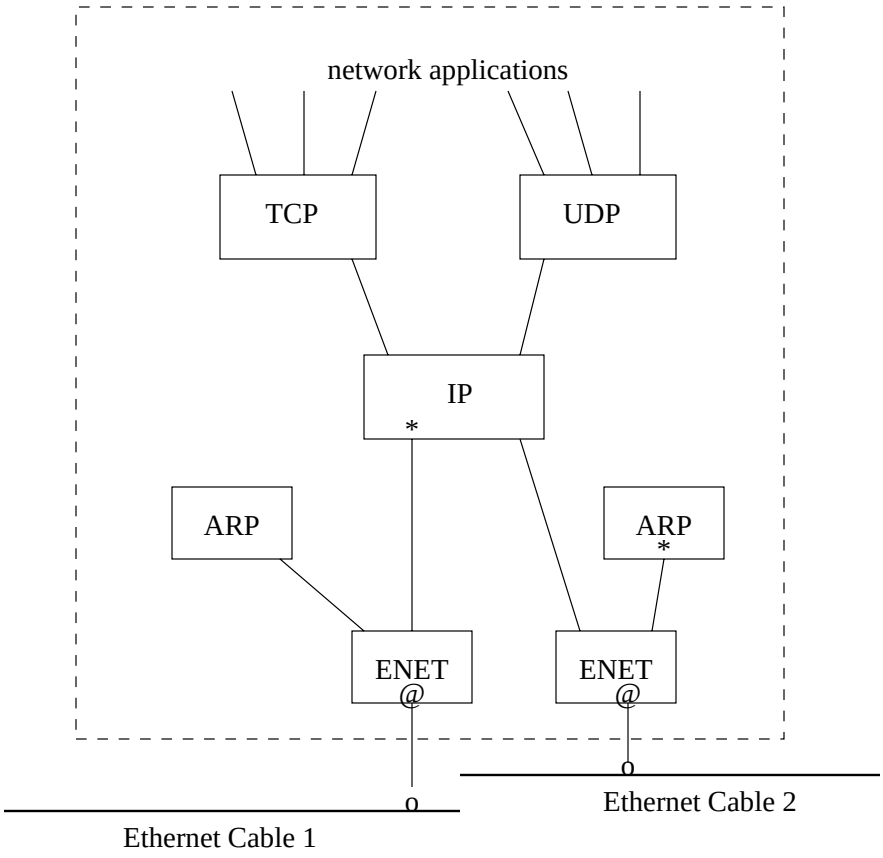


Figure A.3. TCP/IP Network Node on 2 Ethernets

the IP module is both a n-to-m multiplexer and an m-to-n de-multiplexer.

It performs this multiplexing in either direction to accommodate incoming and outgoing data. An IP module with more than 1 network interface is more complex than our original example in that it can forward data onto the next network. Data can arrive on any network interface and be sent out on any other.

The process of sending an IP packet out onto another network is called “forwarding” an IP packet. A computer that has been dedicated to the task of forwarding IP packets is called an

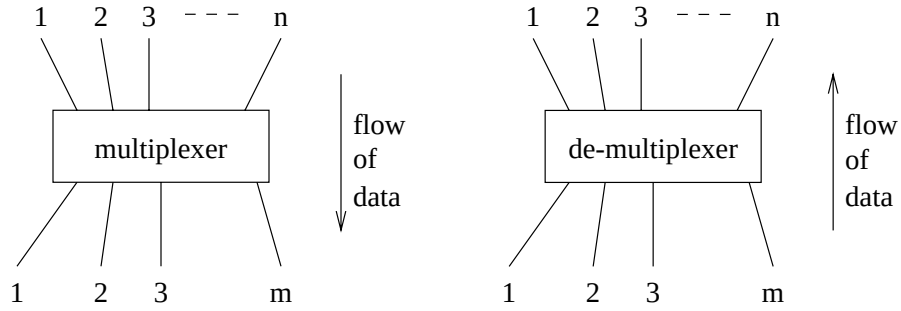


Figure A.4. n-to-m multiplexer and m-to-n de-multiplexer

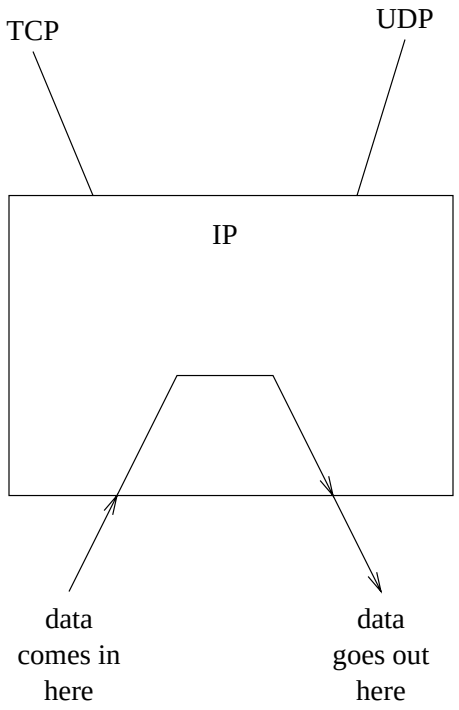


Figure A.5. Example of IP Forwarding a IP Packet

"IP-router".

As you can see from the figure, the forwarded IP packet never touches the TCP and UDP modules on the IP-router. Some IP-router implementations do not have a TCP or UDP module.

A.2.5. IP Creates a Single Logical Network

The IP module is central to the success of internet technology. Each module or driver adds its header to the message as the message passes down through the protocol stack. Each module or driver strips the corresponding header from the message as the message climbs the protocol stack up towards the application. The IP header contains the IP address, which builds a single logical network from multiple physical networks. This interconnection of physical networks is the source of the name: internet. A set of interconnected physical networks that limit the range

of an IP packet is called an “internet”.

A.2.6. Physical Network Independence

IP hides the underlying network hardware from the network applications. If you invent a new physical network, you can put it into service by implementing a new driver that connects to the internet underneath IP. Thus, the network applications remain intact and are not vulnerable to changes in hardware technology.

A.2.7. Interoperability

If two computers on an internet can communicate, they are said to “interoperate”; if an implementation of internet technology is good, it is said to have “interoperability”. Users of general-purpose computers benefit from the installation of an internet because of the interoperability in computers on the market. Generally, when you buy a computer, it will interoperate. If the computer does not have interoperability, and interoperability can not be added, it occupies a rare and special niche in the market.

A.2.8. After the Overview

With the background set, we will answer the following questions:

When sending out an IP packet, how is the destination Ethernet address determined?

How does IP know which of multiple lower network interfaces to use when sending out an IP packet?

How does a client on one computer reach the server on another?

Why do both TCP and UDP exist, instead of just one or the other?

What network applications are available?

These will be explained, in turn, after an Ethernet refresher.

A.3. Ethernet

This section is a short review of Ethernet technology.

An Ethernet frame contains the destination address, source address, type field, and data.

An Ethernet address is 6 bytes. Every device has its own Ethernet address and listens for Ethernet frames with that destination address. All devices also listen for Ethernet frames with a wild- card destination address of “FF-FF-FF-FF-FF-FF” (in hexadecimal), called a “broadcast” address.

Ethernet uses CSMA/CD (Carrier Sense and Multiple Access with Collision Detection). CSMA/CD means that all devices communicate on a single medium, that only one can transmit at a time, and that they can all receive simultaneously. If 2 devices try to transmit at the same instant, the transmit collision is detected, and both devices wait a random (but short) period before trying to transmit again.

A.3.1. A Human Analogy

A good analogy of Ethernet technology is a group of people talking in a small, completely dark room. In this analogy, the physical network medium is sound waves on air in the room instead of electrical signals on a coaxial cable.

Each person can hear the words when another is talking (Carrier Sense). Everyone in the room has equal capability to talk (Multiple Access), but none of them give lengthy speeches because they are polite. If a person is impolite, he is asked to leave the room (i.e., thrown off the net).

No one talks while another is speaking. But if two people start speaking at the same instant, each of them know this because each hears something they haven't said (Collision Detection). When these two people notice this condition, they wait for a moment, then one begins talking. The other hears the talking and waits for the first to finish before beginning his own speech.

Each person has an unique name (unique Ethernet address) to avoid confusion. Every time one of them talks, he prefaces the message with the name of the person he is talking to and with his own name (Ethernet destination and source address, respectively), i.e., "Hello Jane, this is Jack, ..blah blah blah...". If the sender wants to talk to everyone he might say "everyone" (broadcast address), i.e., "Hello Everyone, this is Jack, ..blah blah blah...".

A.4. ARP

When sending out an IP packet, how is the destination Ethernet address determined?

ARP (Address Resolution Protocol) is used to translate IP addresses to Ethernet addresses. The translation is done only for outgoing IP packets, because this is when the IP header and the Ethernet header are created.

A.4.1. ARP Table for Address Translation

The translation is performed with a table look-up. The table, called the ARP table, is stored in memory and contains a row for each computer. There is a column for IP address and a column for Ethernet address. When translating an IP address to an Ethernet address, the table is searched for a matching IP address. Table A.1 is a simplified ARP table.

The human convention when writing out the 4-byte IP address is each byte in decimal and separating bytes with a period. When writing out the 6-byte Ethernet address, the conventions are each byte in hexadecimal and separating bytes with either a minus sign or a colon.

The ARP table is necessary because the IP address and Ethernet address are selected independently; you can not use an algorithm to translate IP address to Ethernet address. The IP address is selected by the network manager based on the location of the computer on the internet. When the computer is moved to a different part of an internet, its IP address must be changed. The Ethernet address is selected by the manufacturer based on the Ethernet address space licensed by the manufacturer. When the Ethernet hardware interface board changes, the Ethernet address changes.

A.4.2. Typical Translation Scenario

During normal operation a network application, such as TELNET, sends an application message

IP ADDRESS	Ethernet Address
223.1.2.1	08-00-39-00-2F-C3
223.1.2.3	08-00-5A-21-A7-22
223.1.2.4	08-00-10-99-AC-54

Table A.1. Example ARP Table

to TCP, then TCP sends the corresponding TCP message to the IP module. The destination IP address is known by the application, the TCP module, and the IP module. At this point the IP packet has been constructed and is ready to be given to the Ethernet driver, but first the destination Ethernet address must be determined.

The ARP table is used to look-up the destination Ethernet address.

A.4.3. ARP Request/Response Pair

But how does the ARP table get filled in the first place? The answer is that it is filled automatically by ARP on an "as-needed" basis.

Two things happen when the ARP table can not be used to translate an address:

1. An ARP request packet with a broadcast Ethernet address is sent out on the network to every computer.

2. The outgoing IP packet is queued.

Every computer's Ethernet interface receives the broadcast Ethernet frame. Each Ethernet driver examines the Type field in the Ethernet frame and passes the ARP packet to the ARP module. The ARP request packet says "If your IP address matches this target IP address, then please tell me your Ethernet address". An ARP request packet looks something like table A.2.

Each ARP module examines the IP address and if the Target IP address matches its own IP address, it sends a response directly to the source Ethernet address. The ARP response packet says "Yes, that target IP address is mine, let me give you my Ethernet address". An ARP response packet has the sender/target field contents swapped as compared to the request. It looks something like table A.3.

The response is received by the original sender computer. The Ethernet driver looks at the Type field in the Ethernet frame then passes the ARP packet to the ARP module. The ARP module examines the ARP packet and adds the sender's IP and Ethernet addresses to its ARP table.

The updated table now looks like table A.4.

A.4.4. Scenario Continued

The new translation has now been installed automatically in the table, just milli-seconds after it was needed. As you remember from step 2 above, the outgoing IP packet was queued. Next, the IP address to Ethernet address translation is performed by look-up in the ARP table then the Ethernet frame is transmitted on the Ethernet. Therefore, with the new steps 3, 4, and 5, the

Sender IP Address	223.1.2.1
Sender Enet Address	08-00-39-00-2F-C3
Target IP Address	223.1.2.2
Target Enet Address	<blank>

Table A.2. Example ARP Request

Sender IP Address	223.1.2.2
Sender Enet Address	08-00-28-00-38-A9
Target IP Address	223.1.2.1
Target Enet Address	08-00-39-00-2F-C3

Table A.3. Example ARP Response

IP address	Ethernet address
223.1.2.1	08-00-39-00-2F-C3
223.1.2.2	08-00-28-00-38-A9
223.1.2.3	08-00-5A-21-A7-22
223.1.2.4	08-00-10-99-AC-54

Table A.4. ARP table after response

scenario for the sender computer is:

1. An ARP request packet with a broadcast Ethernet address is sent out on the network to every computer.
2. The outgoing IP packet is queued.
3. The ARP response arrives with the IP-to-Ethernet address translation for the ARP table.
4. For the queued IP packet, the ARP table is used to translate the IP address to the Ethernet address.
5. The Ethernet frame is transmitted on the Ethernet.

In summary, when the translation is missing from the ARP table, one IP packet is queued. The translation data is quickly filled in with ARP request/response and the queued IP packet is transmitted.

Each computer has a separate ARP table for each of its Ethernet interfaces. If the target computer does not exist, there will be no ARP response and no entry in the ARP table. IP will discard outgoing IP packets sent to that address. The upper layer protocols can't tell the difference between a broken Ethernet and the absence of a computer with the target IP address.

Some implementations of IP and ARP don't queue the IP packet while waiting for the ARP

response. Instead the IP packet is discarded and the recovery from the IP packet loss is left to the TCP module or the UDP network application. This recovery is performed by time-out and retransmission. The retransmitted message is successfully sent out onto the network because the first copy of the message has already caused the ARP table to be filled.

A.5. Internet Protocol

The IP module is central to internet technology and the essence of IP is its route table. IP uses this in-memory table to make all decisions about routing an IP packet. The content of the route table is defined by the network administrator. Mistakes block communication.

To understand how a route table is used is to understand internetworking. This understanding is necessary for the successful administration and maintenance of an IP network.

The route table is best understood by first having an overview of routing, then learning about IP network addresses, and then looking at the details.

A.5.1. Direct Routing

The figure below is of a tiny internet with 3 computers: A, B, and C. Each computer has the same TCP/IP protocol stack as in Figure 1. Each computer's Ethernet interface has its own Ethernet address. Each computer has an IP address assigned to the IP interface by the network manager, who also has assigned an IP network number to the Ethernet.

When A sends an IP packet to B, the IP header contains A's IP address as the source IP address, and the Ethernet header contains A's Ethernet address as the source Ethernet address. Also, the IP header contains B's IP address as the destination IP address and the Ethernet header contains B's Ethernet address as the destination Ethernet address. This is shown in table A.5.

For this simple case, IP is overhead because the IP adds little to the service offered by Ethernet. However, IP does add cost: the extra CPU processing and network bandwidth to generate, transmit, and parse the IP header.

When B's IP module receives the IP packet from A, it checks the destination IP address against its own, looking for a match, then it passes the datagram to the upper-level protocol.

This communication between A and B uses direct routing.

A.5.2. Indirect Routing

Figure A.7 is a more realistic view of an internet. It is composed of 3 Ethernets and 3 IP networks connected by an IP-router called computer D. Each IP network has 4 computers; each computer has its own IP address and Ethernet address.

Except for computer D, each computer has a TCP/IP protocol stack like that in Figure A.1. Computer D is the IP-router; it is connected to all 3 networks and therefore has 3 IP addresses and 3 Ethernet addresses. Computer D has a TCP/IP protocol stack similar to that in Figure A.3, except that it has 3 ARP modules and 3 Ethernet drivers instead of 2. Please note that computer D has only one IP module.

The network manager has assigned a unique number, called an IP network number, to each of the Ethernets. The IP network numbers are not shown in this diagram, just the network names.

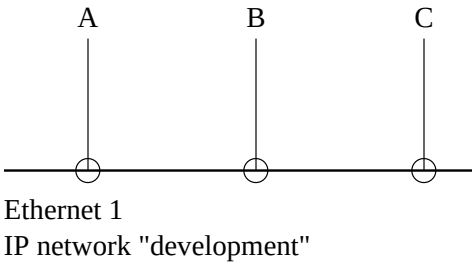


Figure A.6. One IP Network

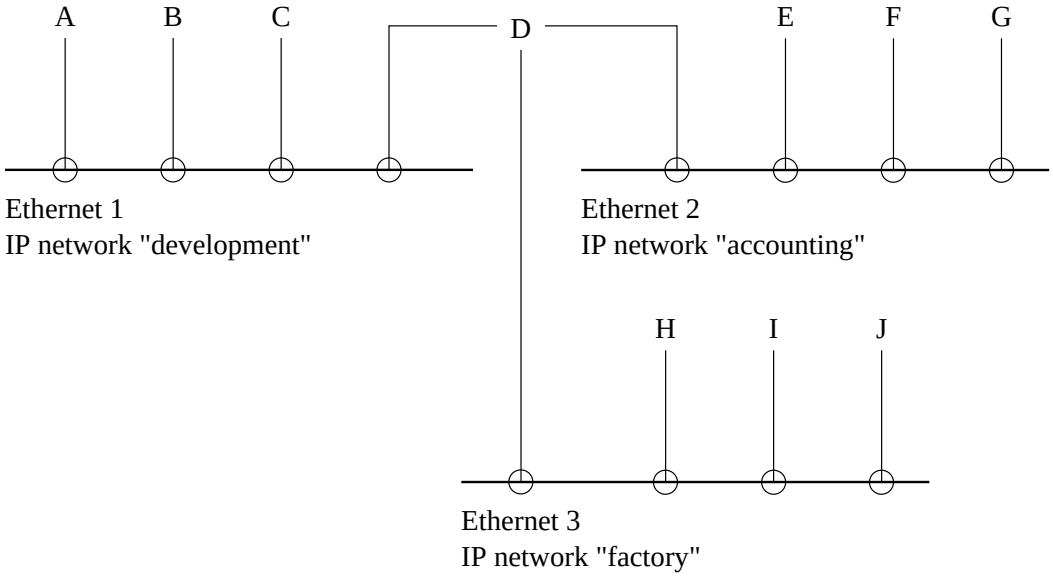


Figure A.7. Three IP Networks; One internet

address	source	destination
IP header	A	B
Ethernet header	A	B

Table A.5. Addresses in an Ethernet frame for an IP packet from A to B

When computer A sends an IP packet to computer B, the process is identical to the single network example above. Any communication between computers located on a single IP network matches the direct routing example discussed previously.

When computer D and A communicate, it is direct communication. When computer D and E communicate, it is direct communication. When computer D and H communicate, it is direct communication. This is because each of these pairs of computers is on the same IP network.

However, when computer A communicates with a computer on the far side of the IP-router, communication is no longer direct. A must use D to forward the IP packet to the next IP network. This communication is called “indirect”.

This routing of IP packets is done by IP modules and happens transparently to TCP, UDP, and the network applications.

If A sends an IP packet to E, the source IP address and the source Ethernet address are A's. The destination IP address is E's, but because A's IP module sends the IP packet to D for forwarding, the destination Ethernet address is D's as shown in table A.6.

D's IP module receives the IP packet and upon examining the destination IP address, says "This is not my IP address," and sends the IP packet directly to E as shown in table A.7.

In summary, for direct communication, both the source IP address and the source Ethernet address is the sender's, and the destination IP address and the destination Ethernet address is the recipient's. For indirect communication, the IP address and Ethernet addresses do not pair up in this way.

This example internet is a very simple one. Real networks are often complicated by many factors, resulting in multiple IP-routers and several types of physical networks. This example internet might have come about because the network manager wanted to split a large Ethernet in order to localize Ethernet broadcast traffic.

A.5.3. IP Module Routing Rules

This overview of routing has shown what happens, but not how it happens. Now let's examine the rules, or algorithm, used by the IP module.

- For an outgoing IP packet, entering IP from an upper layer, IP must decide whether to send the IP packet directly or indirectly, and IP must choose a lower network interface. These choices are made by consulting the route table.
- For an incoming IP packet, entering IP from a lower interface, IP must decide whether to forward the IP packet or pass it to an upper layer. If the IP packet is being forwarded, it is treated as an outgoing IP packet.
- When an incoming IP packet arrives it is never forwarded back out through the same network interface.

These decisions are made before the IP packet is handed to the lower interface and before the ARP table is consulted.

A.5.4. IP Address

The network manager assigns IP addresses to computers according to the IP network to which the computer is attached. One part of a 4- byte IP address is the IP network number, the other part is the IP computer number (or host number). For the computer in table A.1, with an IP address of 223.1.2.1, the network number is 223.1.2 and the host number is number 1.

The portion of the address that is used for network number and for host number is defined by the upper bits in the 4-byte address. All example IP addresses in this tutorial are of type class C, meaning that the upper 3 bits indicate that 21 bits are the network number and 8 bits are the host number. This allows 2,097,152 class C networks up to 254 hosts on each network.

address	source	destination
IP header	A	E
Ethernet header	A	D

Table A.6. Addresses in an Ethernet frame for an IP packet from A to E (before D)

address	source	destination
IP header	A	E
Ethernet header	D	E

Table A.7. Addresses in an Ethernet frame for an IP packet from A to E (after D)

The IP address space is administered by the NIC (Network Information Center). All internets that are connected to the single world-wide Internet must use network numbers assigned by the NIC. If you are setting up your own internet and you are not intending to connect it to the Internet, you should still obtain your network numbers from the NIC. If you pick your own number, you run the risk of confusion and chaos in the eventuality that your internet is connected to another internet.

A.5.5. Names

People refer to computers by names, not numbers. A computer called alpha might have the IP address of 223.1.2.1. For small networks, this name-to-address translation data is often kept on each computer in the "hosts" file. For larger networks, this translation data file is stored on a server and accessed across the network when needed. A few lines from that file might look like this:

```
223.1.2.1   alpha
223.1.2.2   beta
223.1.2.3   gamma
223.1.2.4   delta
223.1.3.2   epsilon
223.1.4.2   iota
```

The IP address is the first column and the computer name is the second column.

In most cases, you can install identical "hosts" files on all computers. You may notice that "delta" has only one entry in this file even though it has 3 IP addresses. Delta can be reached with any of its IP addresses; it does not matter which one is used. When delta receives an IP packet and looks at the destination address, it will recognize any of its own IP addresses.

IP networks are also given names. If you have 3 IP networks, your "networks" file for documenting these names might look something like this:

```
223.1.2    development
223.1.3    accounting
223.1.4    factory
```

The IP network number is in the first column and its name is in the second column.

From this example you can see that alpha is computer number 1 on the development network, beta is computer number 2 on the development network and so on. You might also say that alpha is development.1, Beta is development.2, and so on.

The above hosts file is adequate for the users, but the network manager will probably replace the line for delta with:

```
223.1.2.4    devnetrouter delta
223.1.3.1    facnetrouter
223.1.4.1    accnetrouter
```

These three new lines for the hosts file give each of delta's IP addresses a meaningful name. In fact, the first IP address listed has 2 names; "delta" and "devnetrouter" are synonyms. In practice "delta" is the general-purpose name of the computer and the other 3 names are only used when administering the IP route table.

These files are used by network administration commands and network applications to provide meaningful names. They are not required for operation of an internet, but they do make it easier for us.

A.5.6. IP Route Table

How does IP know which lower network interface to use when sending out a IP packet? IP looks it up in the route table using a search key of the IP network number extracted from the IP destination address.

The route table contains one row for each route. The primary columns in the route table are: IP network number, direct/indirect flag, router IP address, and interface number. This table is referred to by IP for each outgoing IP packet.

On most computers the route table can be modified with the "route" command. The content of the route table is defined by the network manager, because the network manager assigns the IP addresses to the computers.

A.5.7. Direct Routing Details

To explain how it is used, let us visit in detail the routing situations we have reviewed previously and as shown in figure A.8.

The route table inside alpha looks like table A.8.

This view can be seen on some UNIX systems with the "netstat -r" command. With this simple network, all computers have identical routing tables.

For discussion, the table is printed again without the network number translated to its network

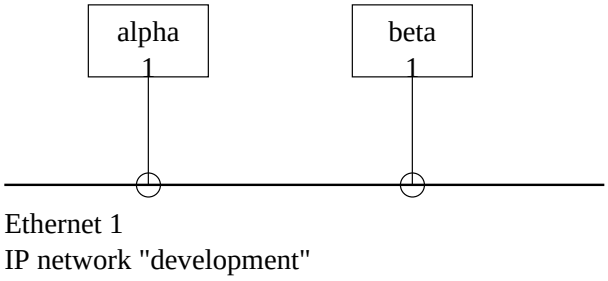


Figure A.8. Close-up View of One IP Network

network	direct/indirect flag	router	interface number
development	direct	<blank>	1

Table A.8. Example Simple Route Table

name as table A.9.

A.5.8. Direct Scenario

Alpha is sending an IP packet to beta. The IP packet is in alpha's IP module and the destination IP address is beta or 223.1.2.2. IP extracts the network portion of this IP address and scans the first column of the table looking for a match. With this network a match is found on the first entry.

The other information in this entry indicates that computers on this network can be reached directly through interface number 1. An ARP table translation is done on beta's IP address then the Ethernet frame is sent directly to beta via interface number 1.

If an application tries to send data to an IP address that is not on the development network, IP will be unable to find a match in the route table. IP then discards the IP packet. Some computers provide a "Network not reachable" error message.

A.5.9. Indirect Routing Details

Now, let's take a closer look at the more complicated routing scenario that we examined previously as shown in figure A.9.

The route table inside alpha looks like table A.10.

For discussion the table is printed again using numbers instead of names in table A.11.

The router in Alpha's route table is the IP address of delta's connection to the development network.

A.5.10. Indirect Scenario

Alpha is sending an IP packet to epsilon. The IP packet is in alpha's IP module and the destination

network	direct/indirect flag	router	interface number
223.1.2	direct	<blank>	1

Table A.9. Example Simple Route Table

network	direct/indirect flag	router	interface number
development	direct	<blank>	1
accounting	indirect	devnetrouter	1
factory	indirect	devnetrouter	1

Table A.10. Alpha Route Table

network	direct/indirect flag	router	interface number
223.1.2	direct	<blank>	1
223.1.3	indirect	223.1.2.4	1
223.1.4	indirect	223.1.2.4	1

Table A.11. Alpha Route Table with Numbers

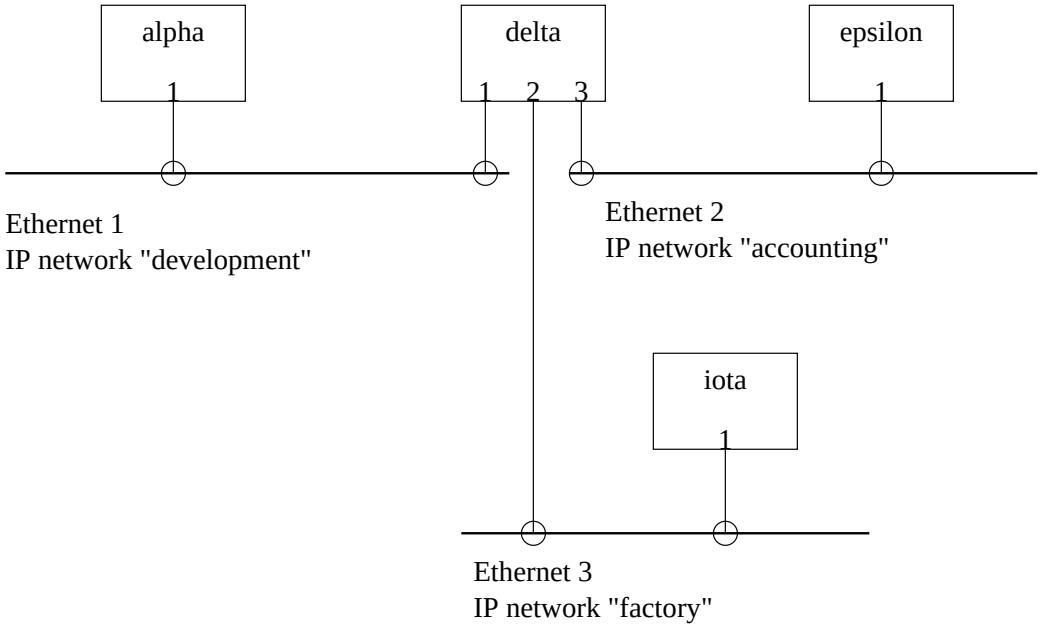


Figure A.9. Close-up View of Three IP Networks

IP address is epsilon (223.1.3.2). IP extracts the network portion of this IP address (223.1.3) and scans the first column of the table looking for a match. A match is found on the second entry.

This entry indicates that computers on the 223.1.3 network can be reached through the IP-router devnetrouter. Alpha's IP module then does an ARP table translation for devnetrouter's IP address and sends the IP packet directly to devnetrouter through Alpha's interface number 1. The IP packet still contains the destination address of epsilon.

The IP packet arrives at delta's development network interface and is passed up to delta's IP module. The destination IP address is examined and because it does not match any of delta's own IP addresses, delta decides to forward the IP packet.

Delta's IP module extracts the network portion of the destination IP address (223.1.3) and scans its route table for a matching network field. Delta's route table looks like table A.12.

Table A.13 is delta's table printed again, without the translation to names.

The match is found on the second entry. IP then sends the IP packet directly to epsilon through interface number 3. The IP packet contains the IP destination address of epsilon and the Ethernet destination address of epsilon.

The IP packet arrives at epsilon and is passed up to epsilon's IP module. The destination IP address is examined and found to match with epsilon's IP address, so the IP packet is passed to the upper protocol layer.

A.5.11. Routing Summary

When a IP packet travels through a large internet it may go through many IP-routers before it reaches its destination. The path it takes is not determined by a central source but is a result of consulting each of the routing tables used in the journey. Each computer defines only the next hop in the journey and relies on that computer to send the IP packet on its way.

A.5.12. Managing the Routes

Maintaining correct routing tables on all computers in a large internet is a difficult task; network configuration is being modified constantly by the network managers to meet changing needs. Mistakes in routing tables can block communication in ways that are excruciatingly tedious to diagnose.

Keeping a simple network configuration goes a long way towards making a reliable internet. For instance, the most straightforward method of assigning IP networks to Ethernet is to assign a single IP network number to each Ethernet.

Help is also available from certain protocols and network applications. ICMP (Internet Control Message Protocol) can report some routing problems. For small networks the route table is filled manually on each computer by the network administrator. For larger networks the network administrator automates this manual operation with a routing protocol to distribute routes throughout a network.

When a computer is moved from one IP network to another, its IP address must change. When a computer is removed from an IP network its old address becomes invalid. These changes require frequent updates to the "hosts" file. This flat file can become difficult to maintain for even medium-size networks. The Domain Name System helps

network	direct/indirect flag	router	interface number
development	direct	<blank>	1
factory	direct	<blank>	3
accounting	direct	<blank>	2

Table A.12. Delta's Route Table

network	direct/indirect flag	router	interface number
223.1.2	direct	<blank>	1
223.1.3	direct	<blank>	3
223.1.4	direct	<blank>	2

Table A.13. Delta's Route Table with Numbers

A.6. User Datagram Protocol

UDP is one of the two main protocols to reside on top of IP. It offers service to the user's network applications. Example network applications that use UDP are: Network File System (NFS) and Simple Network Management Protocol (SNMP). The service is little more than an interface to IP.

UDP is a connectionless datagram delivery service that does not guarantee delivery. UDP does not maintain an end-to-end connection with the remote UDP module; it merely pushes the datagram out on the net and accepts incoming datagrams off the net.

UDP adds two values to what is provided by IP. One is the multiplexing of information between applications based on port number. The other is a checksum to check the integrity of the data.

A.6.1. Ports

How does a client on one computer reach the server on another?

The path of communication between an application and UDP is through UDP ports. These ports are numbered, beginning with zero. An application that is offering service (the server) waits for messages to come in on a specific port dedicated to that service. The server waits patiently for any client to request service.

For instance, the SNMP server, called an SNMP agent, always waits on port 161. There can be only one SNMP agent per computer because there is only one UDP port number 161. This port number is well known; it is a fixed number, an internet assigned number. If an SNMP client wants service, it sends its request to port number 161 of UDP on the destination computer.

When an application sends data out through UDP it arrives at the far end as a single unit. For example, if an application does 5 writes to the UDP port, the application at the far end will do 5 reads from the UDP port. Also, the size of each write matches the size of each read.

UDP preserves the message boundary defined by the application. It never joins two application messages together, or divides a single application message into parts.

A.6.2. Checksum

An incoming IP packet with an IP header type field indicating "UDP" is passed up to the UDP module by IP. When the UDP module receives the UDP datagram from IP it examines the UDP checksum. If the checksum is zero, it means that checksum was not calculated by the sender and can be ignored. Thus the sending computer's UDP module may or may not generate checksums. If Ethernet is the only network between the 2 UDP modules communicating, then you may not need checksumming. However, it is recommended that checksum generation always be enabled because at some point in the future a route table change may send the data across less reliable media.

If the checksum is valid (or zero), the destination port number is examined and if an application is bound to that port, an application message is queued for the application to read. Otherwise the UDP datagram is discarded. If the incoming UDP datagrams arrive faster than the application can read them and if the queue fills to a maximum value, UDP datagrams are discarded by UDP. UDP will continue to discard UDP datagrams until there is space in the queue.

A.7. Transmission Control Protocol

TCP provides a different service than UDP. TCP offers a connection- oriented byte stream, instead of a connectionless datagram delivery service. TCP guarantees delivery, whereas UDP does not.

TCP is used by network applications that require guaranteed delivery and cannot be bothered with doing time-outs and retransmissions. The two most typical network applications that use TCP are File Transfer Protocol (FTP) and the TELNET. Other popular TCP network applications include X-Window System, rcp (remote copy), and the r- series commands. TCP's greater capability is not without cost: it requires more CPU and network bandwidth. The internals of the TCP module are much more complicated than those in a UDP module.

Similar to UDP, network applications connect to TCP ports. Well- defined port numbers are dedicated to specific applications. For instance, the TELNET server uses port number 23. The TELNET client can find the server simply by connecting to port 23 of TCP on the specified computer.

When the application first starts using TCP, the TCP module on the client's computer and the TCP module on the server's computer start communicating with each other. These two end-point TCP modules contain state information that defines a virtual circuit. This virtual circuit consumes resources in both TCP end-points. The virtual circuit is full duplex; data can go in both directions simultaneously. The application writes data to the TCP port, the data traverses the network and is read by the application at the far end.

TCP packetizes the byte stream at will; it does not retain the boundaries between writes. For example, if an application does 5 writes to the TCP port, the application at the far end might do 10 reads to get all the data. Or it might get all the data with a single read. There is no correlation between the number and size of writes at one end to the number and size of reads at the other end.

TCP is a sliding window protocol with time-out and retransmits. Outgoing data must be acknowledged by the far-end TCP. Acknowledgements can be piggybacked on data. Both receiving ends can flow control the far end, thus preventing a buffer overrun.

As with all sliding window protocols, the protocol has a window size. The window size determines the amount of data that can be transmitted before an acknowledgement is required. For TCP, this amount is not a number of TCP segments but a number of bytes.

A.8. Network Applications

Why do both TCP and UDP exist, instead of just one or the other?

They supply different services. Most applications are implemented to use only one or the other. You, the programmer, choose the protocol that best meets your needs. If you need a reliable stream delivery service, TCP might be best. If you need a datagram service, UDP might be best. If you need efficiency over long-haul circuits, TCP might be best. If you need efficiency over fast networks with short latency, UDP might be best. If your needs do not fall nicely into these categories, then the "best" choice is unclear. However, applications can make up for deficiencies in the choice. For instance if you choose UDP and you need reliability, then the application must provide reliability. If you choose TCP and you need a record oriented service, then the application must insert markers in the byte stream to delimit records.

What network applications are available?

There are far too many to list. The number is growing continually. Some of the applications have existed since the beginning of internet technology: TELNET and FTP. Others are relatively new: X-Windows and SNMP. The following is a brief description of the applications mentioned in this tutorial.

A.8.1. TELNET

TELNET provides a remote login capability on TCP. The operation and appearance is similar to keyboard dialing through a telephone switch. On the command line the user types "telnet delta" and receives a login prompt from the computer called "delta".

TELNET works well; it is an old application and has widespread interoperability. Implementations of TELNET usually work between different operating systems. For instance, a TELNET client may be on VAX/VMS and the server on UNIX System V.

A.8.2. FTP

File Transfer Protocol (FTP), as old as TELNET, also uses TCP and has widespread interoperability. The operation and appearance is as if you TELNETed to the remote computer. But instead of typing your usual commands, you have to make do with a short list of commands for directory listings and the like. FTP commands allow you to copy files between computers.

A.8.3. rsh

Remote shell (rsh or remsh) is one of an entire family of remote UNIX style commands. The UNIX copy command, cp, becomes rcp. The UNIX "who is logged in" command, who, becomes

rwho. The list continues and is referred to collectively to as the "r" series commands or the "r*" (r star) commands.

The r* commands mainly work between UNIX systems and are designed for interaction between trusted hosts. Little consideration is given to security, but they provide a convenient user environment.

To execute the "cc file.c" command on a remote computer called delta, type "rsh delta cc file.c". To copy the "file.c" file to delta, type "rcp file.c delta:". To login to delta, type "rlogin delta", and if you administered the computers in a certain way, you will not be challenged with a password prompt.

A.8.4. NFS

Network File System, first developed by Sun Microsystems Inc, uses UDP and is excellent for mounting UNIX file systems on multiple computers. A diskless workstation can access its server's hard disk as if the disk were local to the workstation. A single disk copy of a database on mainframe "alpha" can also be used by mainframe "beta" if the database's file system is NFS mounted on "beta".

NFS adds significant load to a network and has poor utility across slow links, but the benefits are strong. The NFS client is implemented in the kernel, allowing all applications and commands to use the NFS mounted disk as if it were local disk.

A.8.5. SNMP

Simple Network Management Protocol (SNMP) uses UDP and is designed for use by central network management stations. It is a well known fact that if given enough data, a network manager can detect and diagnose network problems. The central station uses SNMP to collect this data from other computers on the network. SNMP defines the format for the data; it is left to the central station or network manager to interpret the data.

A.8.6. X-Window

The X Window System uses the X Window protocol on TCP to draw windows on a workstation's bitmap display. X Window is much more than a utility for drawing windows; it is entire philosophy for designing a user interface.

A.9. Other Information

Much information about internet technology was not included in this tutorial. This section lists information that is considered the next level of detail for the reader who wishes to learn more.

- administration commands: arp, route, and netstat
- ARP: permanent entry, publish entry, time-out entry, spoofing
- IP route table: host entry, default gateway, subnets

- IP: time-to-live counter, fragmentation, ICMP
- RIP, routing loops
- Domain Name System

A.10. References

[1] Comer, D., "Internetworking with TCP/IP Principles, Protocols, and Architecture", Prentice Hall, Englewood Cliffs, New Jersey, U.S.A., 1988.

[2] Feinler, E., et al, DDN Protocol Handbook, Volume 2 and 3, DDN Network Information Center, SRI International, 333 Ravenswood Avenue, Room EJ291, Menlow Park, California, U.S.A., 1985.

[3] Spider Systems, Ltd., "Packets and Protocols", Spider Systems Ltd., Stanwell Street, Edinburgh, U.K. EH6 5NG, 1990.

A.11. Relation to other RFCs

This RFC is a tutorial and it does not UPDATE or OBSOLETE any other RFC.

A.12. Security Considerations

There are security considerations within the TCP/IP protocol suite. To some people these considerations are serious problems, to others they are not; it depends on the user requirements.

This tutorial does not discuss these issues, but if you want to learn more you should start with the topic of ARP-spoofing, then use the "Security Considerations" section of RFC 1122 to lead you to more information.

A.13. Authors' Addresses

Theodore John Socolofsky
Spider Systems Limited
Spider Park
Stanwell Street
Edinburgh EH6 5NG
United Kingdom

Phone:
from UK 031-554-9424
from USA 011-44-31-554-9424
Fax:

from UK 031-554-0649
from USA 011-44-31-554-0649

EMail: TEDS@SPIDER.CO.UK

Claudia Jeanne Kale
12 Gosford Place
Edinburgh EH6 4BJ
United Kingdom

Phone:
from UK 031-554-7432
from USA 011-44-31-554-7432

EMail: CLAUDIAK@SPIDER.CO.UK

Index