EE 706 : Communication Networks
Lecture Notes Version 0.1

Saravanan Vijayakumaran
Department of Electrical Engineering
Indian Institute of Technology, Bombay

Spring 2010

Chapter 1

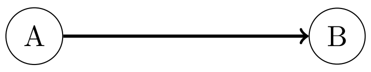
Introduction

1.1 What is a Communication Network?

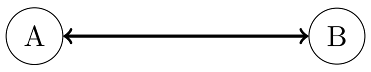
In the context of this course, *communication* is the transfer of information between geographically separated points. The information transferred can be *digital* like a string of bits constituting an image file or it can be *analog* like a voice signal. The systems which are used to achieve communication are called *communication systems*. Communication between two communication systems can either be in one direction or in both directions.

When communication is possible between two communication systems, they are said to be *connected* and a *connection* or a *communication link* is said to exist between them. A *communication network* is a collection of communication systems and the communication links which connect them.

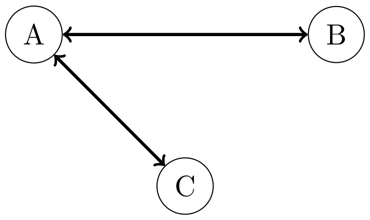
Communication networks are typically illustrated using graphs where the vertices represent communication systems and an edge between a pair of vertices represents a direct connection between the corresponding communication systems. Two communication systems are said to have a direct connection between them if they can communicate without the help of a third communication system. The directionality of the edge denotes the direction in which the communication can take place. The vertices of the graph representing a communication network are commonly referred to as *nodes*. In the interest of brevity, we will use often use the terms nodes and communication systems interchangeably. Figure 1.1a shows a graphical illustration of a communication network consisting of two nodes A and B where the transfer of information can occur only from A to B but not in the other direction. Figure 1.1b shows a communication network again consisting of two nodes where information transfer can occur from either node to the other. These two graphs represent two different communication networks because the connection between the nodes is different. Figure 1.1c shows a communication network with three nodes A, B and C where there is no direct connection between nodes B and C. Information transfer between nodes B and C can occur but this will require the node A to relay the information, which is why we have not drawn an edge between B and C.



(a) Communication network with two nodes connected by a unidirectional link



(b) Communication network with two nodes connected by a bidirectional link



(c) Communication network with three nodes

Figure 1.1: Graphical representation of communication networks

1.2 Issues in Communication Network Design

While the above definition of a communication network is general enough to encompass the postal system, a newspaper distribution system, the announcement sytem at a railway station and FM radio, we will be focussing our attention on computer communication networks. In computer networks, the information to be communicated can always be represented by a string of bits and the communication is performed using electrical or electromagnetic signals. As a consequence of the first characteristic, the communication systems constituting such a network are invariably digital communication systems. By a computer, we mean any electronic device capable of storing, transmitting and receiving digital information. For example, a cell phone, a pager or even a satellite can be considered a computer in this context. Another prevalent characteristic of computer communication networks is that very long information bit strings are broken down into smaller bit strings called *frames* or *packets* which are then transferred across the network sequentially. The advantages of using packets will be explained in the next few sections as and when the appropriate context arises. Let us consider the issues involved in designing a computer communication network.

1.2.1 The Physical Channel

Communication between geographically separated locations requires some physical phenomenon to enable the information transfer. The physical medium through which the communication takes place is called a *channel*. For example, when we speak the acoustic channel is carrying the sound waves generated to the listener.

In the case of computer communications, the channel is chosen based on user requirements and system parameters. For example, if the distance of between the communication systems is small and if their locations are fixed, a conducting wire is used to physically connect them and electrical signals are used to transfer information between them. Such a channel is called a *wired channel*. A fibre optic channel where the information is transferred using optical pulses travelling along an optical fibre is also an example of a wired channel. Even if the distances involved are short but the communication systems are required to be mobile the information is transferred using electromagnetic waves radiated through free space. Such a channel is called a *wireless channel*. A wireless channel is also chosen for long-distance communication between fixed locations because

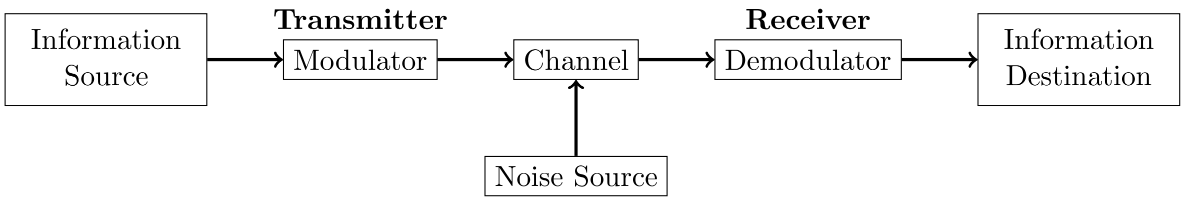


Figure 1.2: Basic block diagram of the communication process

it is more cost-effective than using a wired channel. Sometimes there is no option in choosing between wired and wireless channels like, for example, in satellite communication.

As mentioned before, the information to be transferred in computer communications is a string of bits. Irrespective of whether the channel is wired or wireless, a string of bits is not suitable for immediate sending across a channel. It has to be first mapped to a signal (electric/electromagnetic) which can pass through the channel. This mapping process is called *modulation* and the subsystem of the communication system which performs this mapping is called the *transmitter*. The transmitted signal passes through the channel and arrives at the destination after being subjected to delay, distortion and attenuation. All the distorting effects of the channel are together characterized as *noise*. The communication system at the destination maps the noisy signal back to a string of bits which is ideally a copy of the information at the source. This inverse mapping process is called *demodulation* and the subsystem of the communication system which performs this is called the *receiver*. The demodulation process is not perfect and often results in *errors* which are defined as differences between the string of bits used as input to the modulator and the string of bits obtained as the output of the demodulation process. Since most computer communication systems send and receive information, they contain both the transmitter and receiver subsystems which together constitute what is often called a *modem* (*mo*dulator and *dem*odulator). Figure 1.2 shows a simplified block diagram of the communication process.

The goal of a modem is to transfer information from the transmitter to the receiver as fast as possible with the fewest possible errors. The speed of transfer is called the *data rate* and is simply the number of bits which can be transferred per second. The error performance of a modem is characterized by the *bit error rate (BER)*. The BER is defined as the probability that a bit is received erroneously when it is transmitted across the channel. The maximization of the data rate and minimization of BER are the primary yet conflicting design criteria for modems.

Every physical communication channel has a finite *bandwidth* which is defined as the size of the range of frequencies which pass through the channel without severe attenuation. The noise level in a channel is typically characterized by the *signal-to-noise ratio (SNR)* which is defined as the ratio of the signal power and the noise power in the channel. For a fixed signal power, a channel with high SNR implies a channel with low noise levels and vice versa. The bandwidth together with the SNR in the channel determine the data rates which can be achieved at a particular BER.

The subject of modem design is treated in detail in courses on digital communication. The topics covered include the design of efficient modulation schemes, techniques to combat the harmful effects of channel noise and the design of optimal demodulation schemes. In this course, we will not address these design issues. We will assume that the channels between the nodes in the communication network are capable of transferring bits at a particular data rate and BER. So we are effectively clumping the physical channel and the modem together into a channel which behaves like an unreliable bit pipe. It takes bit strings as input and outputs bit strings which

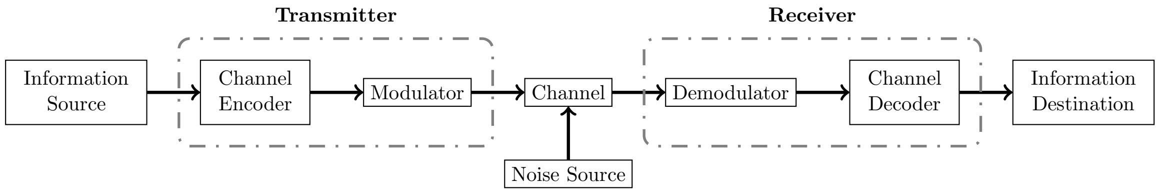


Figure 1.3: Basic block diagram of the communication process including channel encoder and decoder blocks

have some bits flipped relative to the input strings. We will focus our attention on the strategies which are used to enable reliable communication between the nodes of a communication network given that the channels connecting the nodes are unreliable.

1.2.2 Reliable Communication

Most users of computer communication networks cannot tolerate errors irrespective of whether the information being transferred is a text message or a bank account number. Given that the links connecting the nodes in the communication network introduce errors, we need strategies which will eliminate such errors and make the communication reliable or error-free.

Error Correction and Detection

One way to eliminate errors in the received bit strings is to use *forward error correction (FEC)* schemes. An FEC scheme modifies the bit string at the source node before it enters the modulator by adding redundancy to it. This redundancy is used at the receiver to eliminate errors in the bit string obtained as the output of the demodulator at the destination node. An FEC scheme is also called an *error correcting code (ECC)* or *channel code*, where the latter name is due to the fact that the scheme is responsible for correcting errors introduced by the channel distortion. The subsystem of the communication system which adds the redundancy to the input bit string is called the *FEC encoder* or *channel encoder*. The subsystem of the communication system which eliminates errors in the bit string at the output of the demodulator is called the *FEC decoder* or *channel decoder*. Figure 1.3 shows the location of the channel encoder and decoder in the block diagram of the communication process. In this case, the transmitter consists of the channel encoder and the modulator while the receiver consists of the demodulator and the channel decoder.

The number of errors which can be corrected by an ECC is directly proportional to the amount of redundancy which is added to the input bit string. The more the redundancy added, the more the number of errors which can be corrected. But this error correction capability comes at the cost of requiring the tranmission of a bit string which is much longer than the original bit string representing the information to be communicated. The amount of redundancy introduced by an ECC is measured in terms of its *rate* which is defined as the ratio of the lengths of the information bit string and the bit string obtained after the addition of redundancy to information bit string. For example, if the information bit string is k bits long and the bit string obtained after addition of redundancy is $k+n$ bits long, the rate of this ECC will be $\frac{k}{k+n}$. The rate of an ECC is an example of a more general concept called *throughput* which is the defined as the average number of information bits that are communicated through the channel in a unit of time. On a channel with fixed data rate, a lower rate implies a larger delay in transferring the information bit string from source to destination. The tradeoff between rate and error correcting capability is the fundamental challenge in the design of channel codes. Good codes have high values for these two parameters along with low-complexity encoding and decoding algorithms.

For a fixed rate, the number of errors which can be corrected by an ECC is upper bounded. If the number of errors introduced by the channel noise exceeds this upper bound, the channel decoder output will still contain errors. Such a situation is definitely undesirable and we need strategies which can counter it. One strategy is to ask the source node to resend the message. The channel noise is usually a random process and consequently the number of errors it introduces in the transmitted bit string varies from one transmission to the next. So we can hope that the number of errors in the second transmission is less than the number of errors which the ECC can correct, resulting in an error-free communication of the information bit string. This strategy sounds like it might work except for one subtle assumption. It implicitly assumes that we can tell when the channel decoder output contains errors. This is not trivial since the information bit string at the channel encoder input can be any finite length bit string and hence we should expect any bit string of same length to appear at the output of the channel decoder. We can misunderstand a bit string with errors at the channel decoder output to be the error-free decoder output corresponding to a different input bit string. This issue is resolved by preventing the input bit string at the channel encoder output from being any arbitrary bit string. This is done by appending a fixed number of *check bits* to the information bit string which are a deterministic function of the information bit string. At the channel decoder output, the check bits are recalculated using the same deterministic function acting on the decoded information bits and compared to the decoded check bits. The deterministic function is chosen such that errors in the decoder output will cause a discrepancy between the recalculated check bits and the decoded check bits. Such a discrepancy is called an *error detection* and the mapping of the information bit string to a combination of the information bit string and the check bit string is called an *error detection code.*

The simplest example of an error detection code is the single parity check code where a single parity bit is appended to an information bit string. The parity bit is set to 0 if the number of ones in the information bit string is even and to 1 if the number of ones is odd. If there are no errors introduced when this appended bit string is transmitted across the channel, the bits will sum to zero modulo two. If there is a single bit in error, the bits will sum to 1 modulo two and the error is detected. However, this scheme cannot detect two bit errors or any pattern of bit errors of even size for that matter. But it can detect all errors of odd size. In practice, a more powerful error detection code called the *cyclic redundancy check (CRC)* is used. Figure 1.4 shows the location of the CRC encoder and decoder in the block diagram of the communication process. Of course, the CRC code further reduces the throughput of the communication scheme. For example, suppose the CRC code has rate R_1 ($R_1 < 1$) and the ECC has rate R_2 ($R_2 < 1$). Then a communication scheme with data rate D bits per second which only involves the FEC code has throughput $R_2 \times D$ bits per second while a communication scheme which has the same data rate but involves both the CRC and FEC codes has throughput $R_1 \times R_2 \times D$ bits per second which is less than $R_2 \times D$.

Automatic Repeat Request

The process of requesting the source to resend the information once an error is detected in the decoded bit string is called *automatic repeat request (ARQ).* In reality, the destination does nothing when it detects an error in the decoded bit string. However, it does send a reply

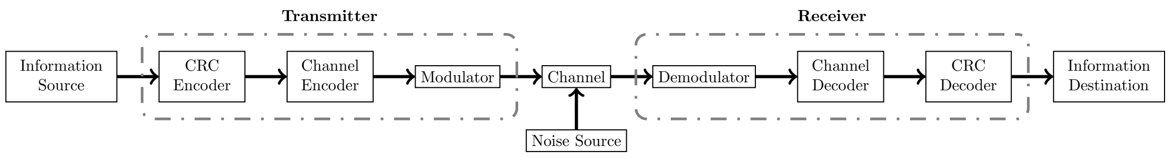


Figure 1.4: Basic block diagram of the communication process including channel code and CRC code blocks

message called an *acknowledgement* when it does not detect any errors in the decoded bit string. The source starts a timer after transmitting a information bit string and waits for an acknowledgement. If the acknowledgement is not received within a predetermined time duration, a *timeout* event occurs and the source decides that the received message was corrupted. It then retransmits the information bit string. Thus ARQ involves implicitly requesting the source to resend the corrupted information.

An error detecting code like the CRC is essential for the correct functioning of an ARQ system but an ECC is not. In fact, an ARQ system is often used to provide reliable communication without using an ECC across channels where bit errors are rare, i.e. when the BER is low. This is because using an ECC results in the transmission of redundant bits even when the channel does not introduce any bit errors, reducing the throughput. On the other hand, using ARQ will result in retransmissions and hence throughput reduction only when the channel introduces bit errors. In this sense, an ARQ-based communication scheme trades throughput for reliability adaptively. The throughput reduction due to the presence of the CRC code occurs irrespective of whether we use an ECC in conjunction with the ARQ scheme or not. Of course, an ECC becomes essential when the channel introduces frequent errors, i.e. when the BER is high. This is because a purely ARQ-based scheme relies on the chance that when an error is detected in the demodulated bit string one of the retransmissions will pass through the channel without any errors being introduced. Such an event is likely only when the BER is low but will require a very large number of retransmissions when the BER is high. In the latter case, using an ECC will result in a reduction in the BER and the subsequent ARQ mechanism will require fewer number of retransmissions.

Now we have enough background to discuss the first advantage of dividing long information bit strings in to smaller bit strings or packets before transmission. Suppose we want to communicate a million bits across a channel which has a BER of 10^{-6}. Such a channel introduces, on the average, one error in every million bits which are transmitted across it. If we transmit the one million bits at once and if an error is detected by the CRC code, we will have to retransmit the million bits. If we assume, for simplicity, that the second transmission contains no bit errors the throughput is halved because we had to transmit twice the number of bits to communicate the million bits. Now consider a packetized system where the million bits are divided into packets of size 100,000 bits each and transmitted one packet at a time. Since the channel introduces one error every million bits, only one of the packets will contain an error and will need to be retransmitted. Once again if we assume the second transmission contains no bit errors, the throughput reduction is only due to the retransmission of the 100,000 bits long packet and is approximately 10%.

In this course, we will discuss several ARQ schemes which represent a tradeoff between throughput, link utilization and buffer size requirements at the transmitter and receiver. The last two parameters in this tradeoff will be introduced and discussed in detail in a later chapter.

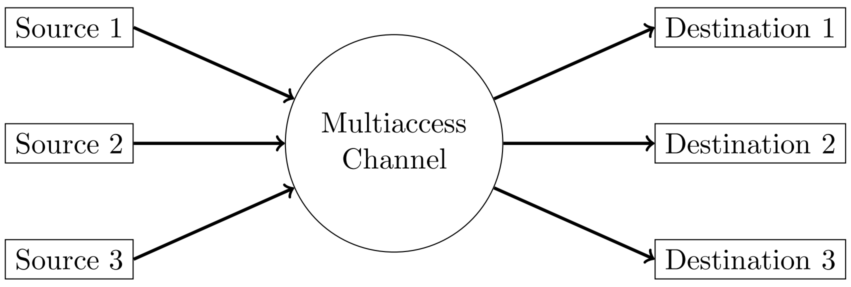


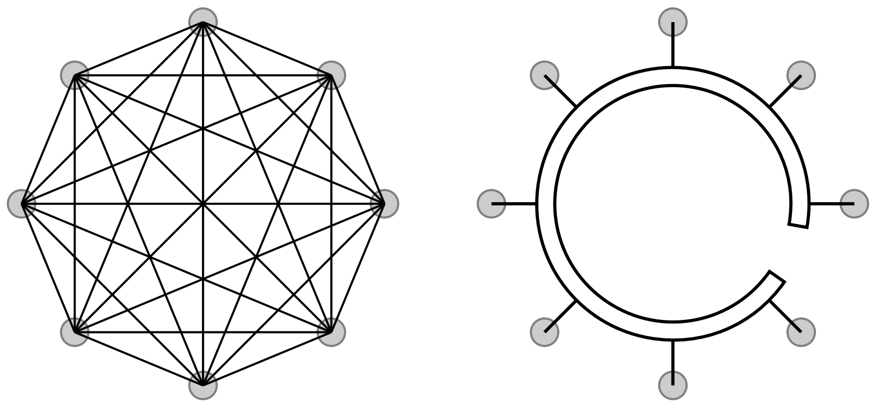
Figure 1.5: Illustration of the multiple access communication process

1.2.3 Multiple Access Channels

The channels we have considered so far are called *point-to-point channels* where the received signal at the destination node depends only on the signal transmitted by the source node and the channel noise. A point-to-point channel is used exclusively by a single source-destination node pair for communication. However, there are some channels called *multiaccess or multiple access channels* which are shared by several source-destination node pairs for communication. In this case, the received signal at a destination node depends on the signal transmitted by several source nodes. Typically, it is the sum of the attenuated transmitted signals corrupted by channel noise. Figure 1.5 illustrates a multiple access communication scenario between three source-destination node pairs. The received signal at any destination depends on the signal transmitted by all three source nodes. The simplest example of a multiple access channel is the wireless channel when multiple source-destination node pairs located within each other's transmission range use the same frequency band to communicate. Multiple access communication does not necessarily imply an equal number of sources and destinations. For example, in satellite communication a single satellite may be communicating with multiple ground stations using the same frequency band.

Multiple access channels are not always wireless. For example, consider a collection of nodes which are located in close proximity to each other. Suppose each node in this collection needs to communicate with every other node. One method to achieve this is to create a point-to-point wired channel from each node to every other node. This is illustrated in Figure 1.6a for a collection of eight nodes where the shaded circles represent the nodes and the edges represent the wired channels. Such an approach does not scale well as the number of nodes increases. A more cost-effective solution which is used in practice is to connect the nodes using a *bus* which is, in this context, a wired multiple access channel as illustrated in Figure 1.6b. Any transmission by a node is sent across the bus and is heard by all the nodes. The node which is the intended receiver will demodulate the received signal and obtain the information destined for it.

There are two issues with the system shown in Figure 1.6b which are common to all multiple access communication systems. The first one is *addressing*. The source node has to include the identity or *address* of the intended receiver of the information along with the information itself. This will enable all the nodes except the intended receiver to ignore the information and the intended receiver to accept it. The address of the receiver or the *destination address* is transmitted before the information bits as part of the *header*. The reason is that the receiving nodes need not process the whole information bit string before ignoring the transmission. Typically the receiving node needs to send back some information back to the source node. To enable this the header also contains the address of the source node or the *source address*. Of course, we have not discussed what exactly constitutes an address. The address structure is highly system



(a) A collection of eight nodes connected by point-to-point wired links

(b) A collection of eight nodes connected by a wired bus

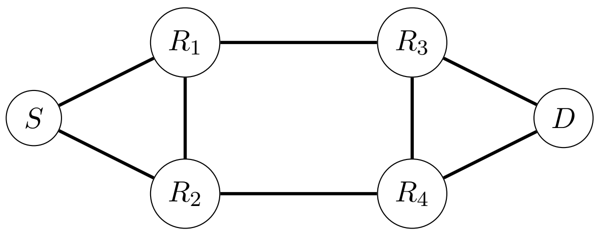
Figure 1.6: Illustration of the point-to-point link and multiple access bus topologies

dependent but in general an address can be thought of as a string of bits which uniquely identify a node. In many systems, the manufacturer assigns this address to a node.

The second and more important issue is that of sharing the channel access among the source-destination pairs or *channel allocation*. A *collision* occurs when two nodes send information at the same time onto the bus. The received signal at all the other nodes will be a superposition of the transmitted signals and cannot be demodulated correctly at their respective receivers. Distributed algorithms called *medium access control (MAC) protocols* are used to avoid or recover from collisions in multiple access channels. For this reason, the addresses described in the previous paragraph are called MAC addresses. MAC protocols require the cooperation of all the nodes sharing the multiple access channel and run in a distributed fashion on all of them.

Some MAC protocols are based on the principle of *random access* which works well if the sources send information infrequently. In this approach, a source node sends information whenever it needs to and hopes for no collisions. When collisions occur, the source node knows that there are other nodes competing to access the channel. All the source nodes involved in the collision then time their retransmissions carefully in order to minimize the chances of future collisions. An improvement over the pure random access MAC protocols is *carrier sense multiple access (CSMA)*. In this scheme, the source node senses the channel to check if a transmission is ongoing and transmits only if the channel is idle. This scheme does not necessarily mean that the transmitted signal needs to have a carrier. It is used in baseband systems as well and the terminology is because it was developed initially for narrowband systems.

Random access MAC protocols do not perform well if the collisions are frequent, that is when the source nodes want to send information frequently. In this case, MAC protocols based on *scheduling or reservation* are used. Such MAC protocols are also called *conflict-free* protocols because they ensure that a transmission is always successful by preventing interference from other transmissions. *Time division multiple access (TDMA)* is a MAC protocol where the time axis is divided into slots and each slot is allotted to a source node for transmission. In *frequency division multiple access (FDMA)*, the frequency bandwidth is divided into smaller bands which are then allocated to the source nodes. The assignment of time slots or frequency bands can be static or dynamic. There are also other reservation-based MAC protocols suitable for wireless channels where the source and destination nodes exchange control messages before actual information transmission. These control messages are received by all the nodes in the transmission range who then refrain from transmitting until the source-destination pair has



(a) A 6-node communication network where every node is reachable from every other node

S routing table		
RN	NH	RC
R_1	R_1	1
R_2	R_2	1
R_3	R_1	2
R_4	R_2	2
D	R_2	3

R_1 routing table		
RN	NH	RC
S	S	1
R_2	R_2	1
R_3	R_3	1
R_4	R_2	2
D	R_3	2

R_2 routing table		
RN	NH	RC
S	S	1
R_1	R_1	1
R_3	R_4	2
R_4	R_4	1
D	R_4	2

R_3 routing table		
RN	NH	RC
S	R_1	2
R_1	R_1	1
R_2	R_4	2
R_4	R_4	1
D	D	1

R_4 routing table		
RN	NH	RC
S	R_2	2
R_1	R_2	2
R_2	R_2	1
R_3	R_3	1
D	D	1

D routing table		
RN	NH	RC
S	R_3	3
R_1	R_3	2
R_2	R_4	2
R_3	R_3	1
R_4	R_4	1

(b) Illustration of routing tables for the 6-node network where RN, NH and RC are abbreviations of reachable node, next hop and routing cost, respectively. The routing cost is the hop count here.

Figure 1.7: Illustration of routing in a communication network

completed communication.

In this course, we will discuss both random access and conflict-free MAC protocols suitable for wired and wireless multiple access channels.

1.2.4 Routing

As the size or geographical spread of a communication network increases, many pairs of nodes will not have a direct communication link and will depend on other nodes to relay information between them. A sequence of relay nodes which transfer information between a source-destination node pair is called a *route*. A route implicitly also consists of the communication links which connect the source and destination to the relay nodes and the links which connects the relay nodes to each other. A node is said to be *reachable* from a source node if a route exists from the source node to it. Figure 1.7a illustrates a communication network where every node is reachable from every other node. One route between S and D can be $S - R_1 - R_3 - D$ which consists of the relay nodes R_1 and R_3. Other possible routes are $S - R_2 - R_4 - D$ and $S - R_1 - R_2 - R_4 - D$.

Discovering a route between a source-destination pair and selecting among multiple routes (if they exist) is the function of a *routing algorithm.* A routing algorithm is a distributed algorithm which is implemented using a *routing table* at each node in the network. The routing table at a source node consists of a list of all nodes reachable from it together with a *next-hop* node and *routing cost* associated with each reachable node. The next-hop node asssociated with a reachable node is the relay node to which the information must be forwarded by the source node in order for the information to be transferred to the reachable node. The next-hop node is always one which is directly connected to the source node. The routing cost is a quantitative measure of the desirabilty or undesirability of a route. Examples of routing cost are the total latency experienced by a packet traversing the route or the *hop count*, the number of links constituting a route. Ideally, we would like the routing table at a node to contain routes which are optimum with respect to the routing cost among all available routes to another node. Figure 1.7b shows routing tables for all the nodes in the network shown in Figure 1.7a using hop count as the routing cost. When node S wants to send information to node D it will look in its routing table (top left table in Figure 1.7b) for the next hop corresponding to D, which is node R_2 in this case. It will send the information to R_2 which will then look for the next hop corresponding to node D in its own routing table (top right table in Figure 1.7b), which is R_4. Node R_2 forwards the information to node R_4 which then looks in its own routing table (bottom middle table in Figure 1.7b) and finds that D is directly connected to it because the next hop corresponding to D is D itself. So R_4 sends the information to D and the information is successfully routed from S to D.

The first issue in routing is the construction of a routing table at a node containing routes to every other node in the network in a distributed manner. This is achieved using *neighbor discovery* and *flooding.* The neighbors of a node in a communication network are defined as all those nodes which have a direct communication link to it. Each node first discovers which nodes are its neighbors and then calculates the routing cost to each of them. It then constructs a routing table containing its immediate neighbors and *floods* it (sends it to all its neighbors). Each node calculates a new routing table based on the information received from its neighbors and floods this information until a stable routing picture of the network is obtained. Implicit in this description is the existence of a unique identity or *routing address* associated with each node. This address is different from the MAC address described in the section on multiple access channels. While MAC addresses only need to be unique, routing addresses need to have some kind of logical structure to keep the routing table sizes small. A simply unique routing address will result in the routing table at every node containing one entry per node. But if the routing addresses are assigned to nodes such that a group of nodes which are located near each other have routing addresses with the same prefix, then the routing table at a node which is far away from this group only needs to have one entry containing the prefix for all the members of this group. A small routing table not only reduces the memory requirement at each node but also reduces the amount of information which needs to be exchanged between the nodes.

The second issue in routing is ensuring that the routes present in the routing table are the optimal routes in the sense of having minimum routing cost. This is achieved by the routing protocols which are implemented such that the route to a node is updated only when a route having lower routing cost is known. This is complicated further by the occurrence of sporadic node or link failures which make some routing table entries invalid or by the appearance of new routes with lower cost which make some routing table entries suboptimal. Thus routing algorithms need to be running continuously to prevent the routing tables from becoming stale.

A consequence of the presence of multiple routes is the possibility of *out-of-order delivery* of packets at a destination, that is packets which were sent in a particular order by the source

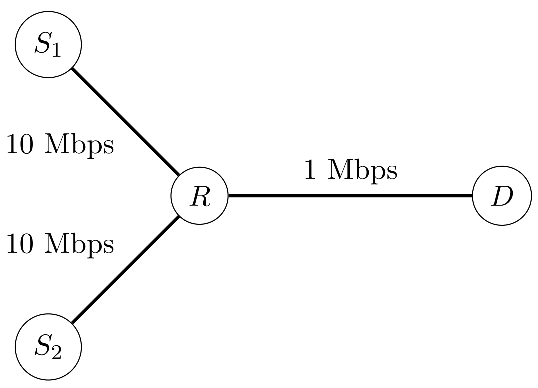


Figure 1.8: Illustration of congestion in a four-node communication network

arrive at the destination in a different order. Hence there is a need to tag the packets sent by the source with a *sequence number* which can then be used by the destination to reorder them. Such a sequence number is also useful in requesting the source to resend a packet in case one is lost and the other arrive correctly.

In this course, we will discuss different routing algorithms and addressing schemes used to achieve routing in the Internet.

1.2.5 Flow and Congestion Control

When a packet arrives at a receiving node, it is stored in a region of memory present in the node which is called a *buffer*. The buffer is used to temporarily store the packet until further processing can be performed on it. This subsequent processing can be the copying of the packet to a disk drive or consumption of the information in the packet by the end user. An example of the latter case is when the information in the packet represents part of a music file which the end user is listening to. Once the necessary processing has been performed on the packet, the corresponding buffer can be freed and reused. The amount of buffer space available in a node is limited. If packets arrive faster than they can be processed, the buffer will be filled at a faster rate than it can be emptied. This will eventually result in the buffer being filled up completely and subsequently arriving packets will be dropped. Such an event is called a *buffer overflow*. This is undesirable since it results in loss of information even when the channel connecting the source and destination is error-free. A mechanism which throttles the rate at which the source sends information to the destination and prevents buffer overflow is called a *flow control* mechanism. Flow control is performed by the destination by informing the source regarding the maximum amount of information it can receive without overflowing its buffer. It also informs the source about the previously sent information which it has already processed and removed from the buffer. The source is then able to calculate the amount of information it can safely send without causing buffer overflow at the destination.

Flow control prevents a single source from overflowing the buffer at a destination. *Congestion control*, on the other hand, prevents a set of sources from causing buffer overflows anywhere in the communication network. A network is said to be *congested* if buffer overflows occur frequently in its nodes. Figure 1.8 shows a four-node network with two sources, one relay node and one destination. The links from the sources S_1 and S_2 to the relay node R support a data rate of ten megabits per second (Mbps). The link from R to the destination D supports a data rate of 1 Mbps. If both the sources send information destined for D at 10 Mbps, the packets will get queued at the relay node R since the outgoing link to D supports information transfer at only 1 Mbps. The buffer at node R will overflow and packets will begin to be dropped at R resulting in congestion. Although the packets are being dropped at R, it is common to say that the links from S_1 to R and S_2 to R are congested because it is the packets arriving on these links which are being dropped.

An obvious solution to the congestion in the above example is to employ flow control on the sources S_1 and S_2 by asking the relay node R to throttle the amount of information they send to it. But this method causes additional packets to be sent from R to the sources which is not scalable as the number of sources increases. A simpler method which does not require the relay node to transmit additional packets is the following. Each source starts a timer waiting for an acknowledgement from the destination D for every successful packet reception. If the packets are dropped at R, timeout events will occur at a source and it will infer that the network is congested. It then reduces the rate at which it transmits information. If both the sources reduce their individual rates of transmission, the congestion will reduce and eventually disappear.

Congestion is considered a serious problem in communication networks and sources often reduce their transmission rates aggressively in order to prevent it as quickly as possible. Once congestion has been avoided, the source transmission rates may be much lower than what the network can support without congestion. Such a situation is also undesirable as it increases the delay in the information transfer from the sources to their respective destinations. The solution is to continue to reduce source transmission rates aggressively when timeouts occur but increase the rates conservatively if acknowledgements arrive from the destination. Such a congestion control scheme guarantees that congestion is prevented quickly and that the source transmission rates are slowly brought up to levels where they do not result in congestion.

This scheme helps the sources recover from congestion once it occurs but it needs some packets to be dropped before it can detect congestion. An alternative is to use schemes which predict the ocurrence of congestion and take steps to avoid it before it occurs. Such schemes are called *congestion avoidance* schemes. A simple example is *random early detection (RED)* where the relay nodes monitor the amount of free buffer space and when they find that it is about to be exhausted they randomly drop a packet which was sent to them. This causes the source of the packet to timeout while waiting for the acknowledgement corresponding to that packet from the destination and eventually reduce its transmission rate. If there are other sources which can still cause congestion at the relay node, their packets will be populating a significant percentage of the buffer and a random drop will pick their packet with high probability. In this way, all the sources can possibly cause congestion will be eventually throttled.

In this course, we will discuss flow control, congestion control and congestion avoidance schemes which are either used or proposed for use on the Internet.

1.2.6 Security

Network security is the field addressing the problem of detecting, preventing or recovering from any action which compromises the security of the information being transferred in a communication network. The actions which constitute a security compromise can be one of the following:

- *Eavesdropping*: The users of a communication network often wish to communicate sensitive information which needs to be kept secret from unauthorized users who are also called *adversaries*. Eavesdropping is said to occur if an adversary can intercept and understand an information transfer in the network. Eavesdropping can be prevented by encrypting the information before communication and a scheme which achieves this is said to provide *confidentiality.*

- *Message modification*: Even when an adversary cannot decode the contents of an information transfer nothing prevents her from modifying the message resulting in a incorrect message being delivered to the destination. A scheme which can detect message modifications is said to provide *data integrity.*
- *Replay attack*: A replay attack occurs when an adversary copies the information sent by the source to the destination and resends it to produce harmful consequences. A scheme which prevents replay attacks is said to provide *originality.*
- *Delaying tactics*: An adversary can also cause harm by intercepting an information transfer from a source and sending it to the destination after a some delay. This does not violate originality but can still have unintended consequences. A scheme which detects delaying tactics is said to provide *timeliness.*
- *Masquerading*: An adversary can claim to be a particular authorized user and initiate communication with other authorized users. A scheme which ensures that users are who they claim to be is said to provide *authentication.* Authentication schemes are used to provide *access control* which is the ability to limit access of network resources to authorized users.
- *Repudiation or bogus denial*: Sometimes a source or a destination can deny that a communication which actually occurred never took place. A scheme which prevents such bogus denials is said to provide *nonrepudiation.*

In this course, we will discuss (if time permits) the major security schemes used on the Internet today.

1.3 Course Goals

- Understand the major issues in the design of communication networks.
- Understand the functioning of existing networks like the Internet, 802.11 WLAN and Ethernet.
- Learn simple analytical tools used to do performance analysis of network protocols.
- Learn to do network simulation.

1.4 Reference Books

- *Computer Networks: A Systems Approach*, Larry Peterson and Bruce Davie, 2007 (4th Edition)
- *Communication Networks: Fundamental Concepts and Key Architectures*, Alberto Leon-Garcia and Indra Widjaja, 2004 (2nd Edition)
- *Computer Networks*, Andrew Tanenbaum, 2002 (4th Edition)

- *Data Networks*, Dimitri Bertsekas and Robert Gallager, 1992 (2nd Edition)

Chapter 2

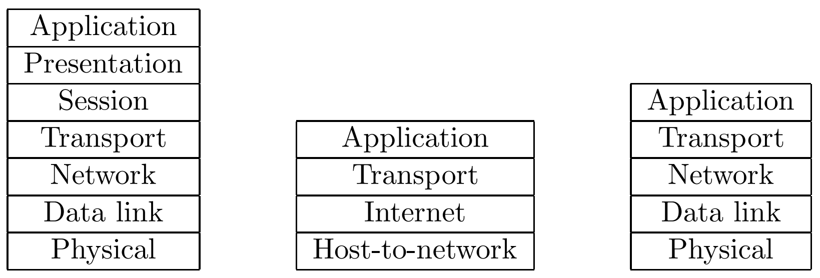
Layering

In the previous chapter, we briefly looked at several issues in communication network design like the physical channel, reliability in the presence of errors, multiple access channels, routing, flow control, congestion control and security. The algorithms or schemes which provide a solution to such issues are called *protocols*. Each of the issues requires the hardware or software implementation of a corresponding protocol. Every node in the communication network needs to contain protocol implementations addressing at least some, if not all, of the issues. The complexity of the implementations can be greatly reduced if the protocols corresponding to different issues are organized in a modular fashion. In all real-world communication networks, the protocols are grouped into *layers* which are arranged in a stack configuration. Each layer is responsible for addressing one or more network issues and the mapping of issues to layers is chosen to enable less complex implementations. Figure 2.1 shows a stack of layers together with the issues each layer addresses. For example, layer 3 is responsible for routing and the protocols which constitute this layer will be routing algorithms. Of course, this figure does not show the protocols which constitute each layer.

The stacking of the layers signifies that each layer interacts only with the layers adjacent to it. Each layer is said to provide a *service* to the layer above it by performing the task assigned to it. The layer above can then operate under the assumption that the layer below it has fulfilled its responsibilities. In Figure 2.1, for example, at a particular node the routing protocol in layer 3 can operate under the assumption that layer 2 has achieved error-free communication to its neighboring nodes. In this sense, the organization of the layers is such that the bottom layers handle issues that are more essential. The ordering of the layers in the stack also represents the order in which transmitted or received information is acted upon by the different layers. When a packet is received from the channel, layer 1 will first process it by demodulating the received signal. Then layer 2 will check if the received packet contains errors and send an acknowledgement to the source if there are no errors. If errors are present, layer 2 will wait for a retransmission from the source and pass the packet to layer 3 only when it does not detect any

Layer 4	Congestion Control, Flow Control
Layer 3	Routing
Layer 2	Reliability, MAC
Layer 1	Modulation, Demodulation

Figure 2.1: An example showing the mapping of communication network issues to layers



(a) The seven-layer OSI model (b) The four-layer TCP/IP model (c) The five-layer hybrid model

Figure 2.2: The OSI, TCP/IP and hybrid reference models

errors in it. Then, layer 3 will check the destination routing address embedded in the packet and forward the packet to the next hop node if the destination routing address does not match the current node's routing address. If there is a match, it will pass the packet to layer 4 which can then perform flow control by throttling the source transmission rate if buffer overflow is imminent. Similarly, when a node has information to send to a destination, layer 4 will first check if it needs to reduce the size of the packet to conform to the flow control restrictions imposed by the destination and then pass the packet to layer 3. Layer 3 will add the routing address of the destination and routing address of the current node to the packet. It will also find the next hop node to reach the destination by going through the entries in the routing table and pass the packet to layer 2. Layer 2 will add FEC and CRC redundancy to the packet and pass it to layer 1 which then modulates the packet bits into signal suitable for transmission over the physical channel.

A collection of layers and the protocols which constitute them is called a *network architecture*. There are two well-known models for a network architecture - *the OSI reference model* and *the TCP/IP reference model*.

2.1 The OSI Reference Model

The ISO OSI (Open Systems Interconnection) reference model was developed to promote the international standardization of the various layers of a network architecture. It is often referred to as just the OSI reference model or OSI model for brevity. It consists of seven layers which are shown in Figure 2.2a. We discuss the responsibilities of each layer below.

- *Physical Layer*: The physical layer is mainly concerned with modulation and demodulation, i.e. transforming information bits received from higher layers to signals suitable for transmission across the physical channel and transforming received signals to bits which can be passed on to higher layers.

- *Data Link Layer*: The data link layer is concerned with making the unreliable bit pipe provided by the physical layer between adjacent nodes into an error-free bit pipe. It achieves this using CRC and ARQ. The data link layer is also concerned with flow control between adjacent nodes. Since adjacent nodes may be connected by a multiple access channel, medium access control is also the responsibility of the data link layer. A sublayer of the data link layer called the MAC sublayer addresses this issue.

- *Network Layer*: Routing packets from source to destination using multiple relay nodes if necessary is the responsibility of the network layer. This implicitly involves the assignment of unique routing addresses to all the nodes in the network. Interconnecting heterogeneous networks (*internetworking*) and congestion control are also the responsibilities of the network layer.

- *Transport Layer*: The transport layer is concerned with delivering a message from a process on the source node to a process on the destination. There may be multiple processes running on a receiving node all of which are expecting to receive packets on a single interface. The transport layer *demultiplexes* the packets arriving at a node to the appropriate process. The layers from the transport layer upwards are all end-to-end layers, that is they are concerned with end-to-end delivery. The three layers below the transport layer are concerned with issues which occur between a node and its immediate neighbors. The transport layer can provide a service corresponding to the end-to-end delivery of packets which are error-free and in order. It can also provide a best-effort service of sending isolated messages which has no guarantees of about the reliability or order of delivery.

- *Session Layer*: The session layer's responsibility is to enable sessions between source and destination. It provides services like dialog control and synchronization between local and remote applications.

- *Presentation Layer*: The presentation layer helps translate between different data representation formats across different machines and networks. The presentation layer prepares the received data into a form which the application layer can understand.

- *Application Layer*: The application layer provides a number of protocols which are used by networking applications. Examples are HTTP (HyperText Transfer Protocol), FTP (File Transfer Protocol) and SMTP (Simple Mail Transfer Protocol).

2.2 The TCP/IP Reference Model

The TCP/IP protocol was born out of a need to interconnect multiple heterogeneous networks seamlessly in the ARPANET, which was a research network sponsored by the U.S. Department of Defense in the 1970s. The TCP/IP reference model was developed much later to conveniently describe the already existing layered structure of the TCP/IP protocol. The TCP/IP reference model consists of only four layers - application, transport, internet and host-to-network layers, as shown in Figure 2.2b. The application layer in the TCP/IP reference model corresponds to the application layer in the OSI reference model. The TCP/IP model does not have the session and presentation layers of the OSI model. If an application needs the functionality which these layers provide under the OSI model, it would have to implement it on its own.

The transport and internet layers correspond to the transport and network layers of the OSI model, respectively. The protocol which enables internetworking in the internet layer is called *IP (Internet Protocol)*. In addition to internetworking, the internet layer is responsible for packet routing and congestion control/avoidance. The transport layer defines two end-to-end transport protocols: *TCP (Transmission Control Protocol)* and *UDP (User Datagram Protocol)*. TCP is a reliable connection-oriented protocol that allows a byte stream originating on one machine to be delivered without error to any other machine on the internet. UDP, on the other hand, is an unreliable, connectionless protocol for applications like streaming audio which inherently can tolerate occassional errors but not the delays which are the cost of reliability.

The TCP/IP reference model does not specify the responsibilities of the layers below the internet layer, except that the host should connect to the network using some protocol so that it can send IP packets.

2.3 A Hybrid Reference Model

The OSI reference model was created to be a guideline for implementing network protocols which conformed to international standards. However, it was not widely adopted due to its complexity and due to the widespread use of the TCP/IP protocols. The TCP/IP reference model was created as an afterthought to describe the existing implementation of the widely used TCP/IP protocols. The downside of this model was that it was not suitable for describing non-TCP/IP networks like Bluetooth.

For pedagogical convenience, a hybrid reference model has been widely adopted in literature which is shown in Figure 2.2c. It essentially looks like the OSI model with the exception of the session and presentation layers. However, the coverage of the transport and network layers in this hybrid model will focus on the TCP/IP protocols which is justified by their widespread use.

In this course, we will use the hybrid reference model to organize related topics.

Chapter 3

Physical Layer

The physical layer is concerned with modulation at a source node and demodulation at the destination node. As we mentioned before, the maximization of the data rate and minimization of the BER are the primary design criteria for the modulation/demodulation subsystems. However, suboptimal demodulation schemes are sometimes used in the interest of reducing the implementation complexity or increasing system robustness. The type of modulation and demodulation techniques used depends of the characteristics of the channel connecting the source and the destination.

A channel is characterized by the range of transmitted signal frequencies it allows to pass through without severe attenuation as well as the type of distortion it induces on the transmitted signal. A channel which acts as a lowpass filter by allowing frequencies from zero upto a maximum value to pass through is called a *baseband channel*. A channel which acts like a bandpass filter by severely attenuating all frequencies except those present in a band located away from the zero frequency is called a *narrowband or bandpass channel*. These two types of channels present different challenges at the physical layer.

In this chapter, we give a brief description of the purpose and functionality of the various physical layer subsystems. The goal is to enable the reader to see where the algorithms described in digital communication courses are used to enable the correct functioning of a communication network.

3.1 Baseband Communication

Baseband channels typically appear when the channel is a conducting wire connecting the source and destination. For example, the physical channel in Ethernet is a baseband channel consisting of a twisted pair of wires. On baseband channels, pulse amplitude modulation (PAM) is the method used to transmit digital information. PAM involves varying the amplitude of a pulse of finite duration according to digital information to be sent. For example, the bit 1 is mapped to the signal $Ap(t)$ and the bit 0 is mapped to the signal $-Ap(t)$ where $p(t)$ is a unit-energy pulse of duration T and $A > 0$ is the amplitude of the pulse. Then a string of bits $\{b_k : k = 0, 1, \dots, N\}$ can be mapped to a sequence of pulses

$$x(t) = \sum_{k=0}^{N} B_k p(t - kT) \tag{3.1}$$

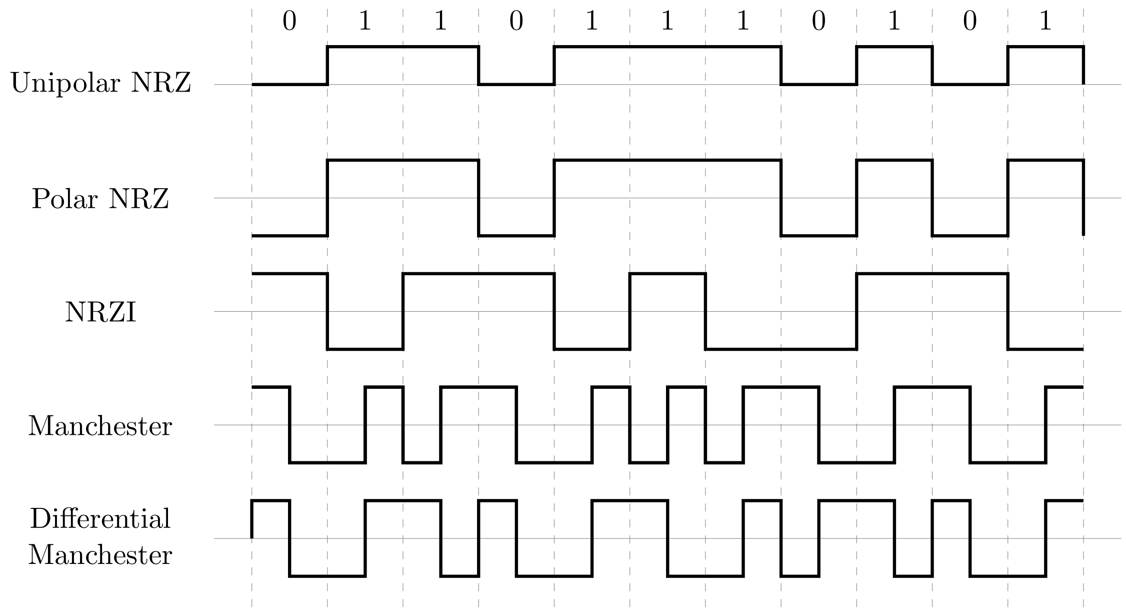


Figure 3.1: Illustration of the different line coding schemes

where $B_k = +A$ if $b_k = 1$ and $B_k = -A$ if $b_k = 0$. This scheme of mapping bit strings to signals is called *polar nonreturn-to-zero (NRZ) encoding* when $p(t) = 1$ for $t \in [0, T]$ and 0 otherwise. The data rate of this scheme is $\frac{1}{T}$ bits per second. A variation of this scheme is the *unipolar NRZ encoding* where the bit 1 is mapped to a positive amplitude and the bit 0 is mapped to the zero amplitude. Such schemes which maps bit strings to pulse trains are called *line coding* schemes and are illustrated in Figure 3.1. Another variation which is an example of a *differential modulation* technique is *NRZ inverted (NRZI) encoding.* In NRZI, the bit 1 is mapped to signal level transition at the beginning of the bit interval and the bit 0 mapped to no signal level transition. In these schemes, a long string of ones or zeroes in the information bit string results in a constant amplitude which can result in the loss of timing information at the receiver. The receiver typically has a timing recovery circuit which uses the transitions at the edge of bit intervals to determine the bit boundaries. The knowledge of the bit boundaries is essential for the correct operation of the matched filter demodulator which is optimum in this scenario. There is always a drift between the clocks running in the transmitter and the receiver, even if they run at the same frequency. The absence of transitions can throw off the receiver's timing recovery circuit. Note that NRZI encoding prevents the lack of transitions when a long string of ones is sent but a long string of zeros will still cause a problem. Another advantage of NRZI is that it will work even if the signal polarities are reversed due to incorrect wiring.

The two main limitations of the NRZ schemes is the presence of a dc component and lack of guaranteed synchronization capability. The *Manchester* encoding scheme, which is used in Ethernet, solves these problems by dividing the bit duration into two halves. The bit 1 is sent by setting the signal to $-A$ during the first half and to $+A$ in the second half. So the bit 1 involves a mid-bit transition from low to high. The bit 0 is sent by having a mid-bit transition from high to low. There is always a mid-bit transition which can aid in timing recovery and there is no dc component. A differential Manchester scheme can be used to work in the case of polarity reversal. It uses the maps a 0 bit to a signal level transition at the beginning of a bit interval and a bit 1 to the absence of a transition at the beginning of a bit interval.

No matter which encoding scheme is used, the pulse duration is finite and a timelimited signal is not bandlimited. So when the pulse is sent through a baseband channel which acts as a lowpass filter, it will get smeared beyond its original duration. When a sequence of amplitude modulated pulses are transmitted through this channel, each pulse will experience *intersymbol interference (ISI)* from its neighboring pulses. The smaller the pulse duration T the more severe the ISI. However, if $T > \frac{1}{2W}$ where W is the cutoff frequency of the baseband channel ISI can be avoided by choosing the transmitted pulses carefully so that the received pulses satisfy the *Nyquist condition* for zero ISI. Thus the channel bandwidth limits the maximum pulse rate $\frac{1}{T}$ to less than $2W$ pulses per second which is called the *Nyquist signaling rate.* If the channel response is not known a priori, making the received pulses satisfy the Nyquist condition is not possible. In this case, the ISI can be cancelled by filtering the received signal. This process is called *equalization.* Equalization involves estimating the channel response and then using this estimate to remove the ISI. To enable channel estimation, the information to be transmitted is preceded by a *preamble* which is predetermined sequence of bits agreed upon by the transmitter and receiver. This sequence is also called the *training sequence.* The training sequence also aids in the initial estimation of the bit boundaries in the received signal. The receiver typically detects the arrival of a packet by matching the training sequence. After the initial estimation of the timing, the signal level transitions induced by the line coding schemes help update the estimate in case of clock drifts. This updating is performed using a *delay-locked loop (DLL)* circuit.

If we send one bit per pulse when the pulse rate is equal to the Nyquist rate, the maximum data rate is limited to $2W$ bits per second. However, we can map multiple bits to a single pulse by allowing more than two levels. For example, we could map 00 to $-Ap(t)$, 01 to $-\frac{A}{3}p(t)$, 10 to $\frac{A}{3}p(t)$ and 11 to $Ap(t)$. In that case, the maximum data rate becomes $4W$ bits per second. In general, if the number of amplitude levels are 2^m where m is a positive integer the maximum data rate is $2Wm$ bits per second. This seems to suggest that we can increase the achievable data rate by increasing the number of levels. But this strategy does not work due to the presence of noise. As the number of levels increase while keeping the maximum amplitude constant, the levels will get closer to each other and can be confused for each other easily resulting in demodulation errors. These ideas are due to Claude Shannon who proved that the maximum data rate in an additive white gaussian noise (AWGN) channel with bandwidth W and SNR S can support with arbitrarily small BER is less than the following value,

$$C = W \log_2(1 + S) \quad \text{bits/second} \tag{3.2}$$

which is called the *capacity* of the channel. For our purposes, we assume that the a finite-bandwidth baseband channel allows us to transmit at a finite data rate with a small but non-zero BER where the data rates are directly proportional to the bandwidth.

In summary, the main physical layer issues in baseband channels are timing recovery and ISI mitigation. Both of them influence the type of modulation scheme which is employed. For example, line coding schemes which have frequent signal level transitions are chosen because they aid timing recovery at the receiver. Transmitted pulse shapes are chosen to minimize ISI at the receiver. So in fact it is the ease of demodulation in the presence of these issues which is the factor determining the choice of modulation scheme. When pulse shaping to avoid ISI is not feasible, equalization is used to remove it. Both equalization and initial timing recovery are accomplished by the presence of a preamble which is sent immediately preceding the information-bearing signal.

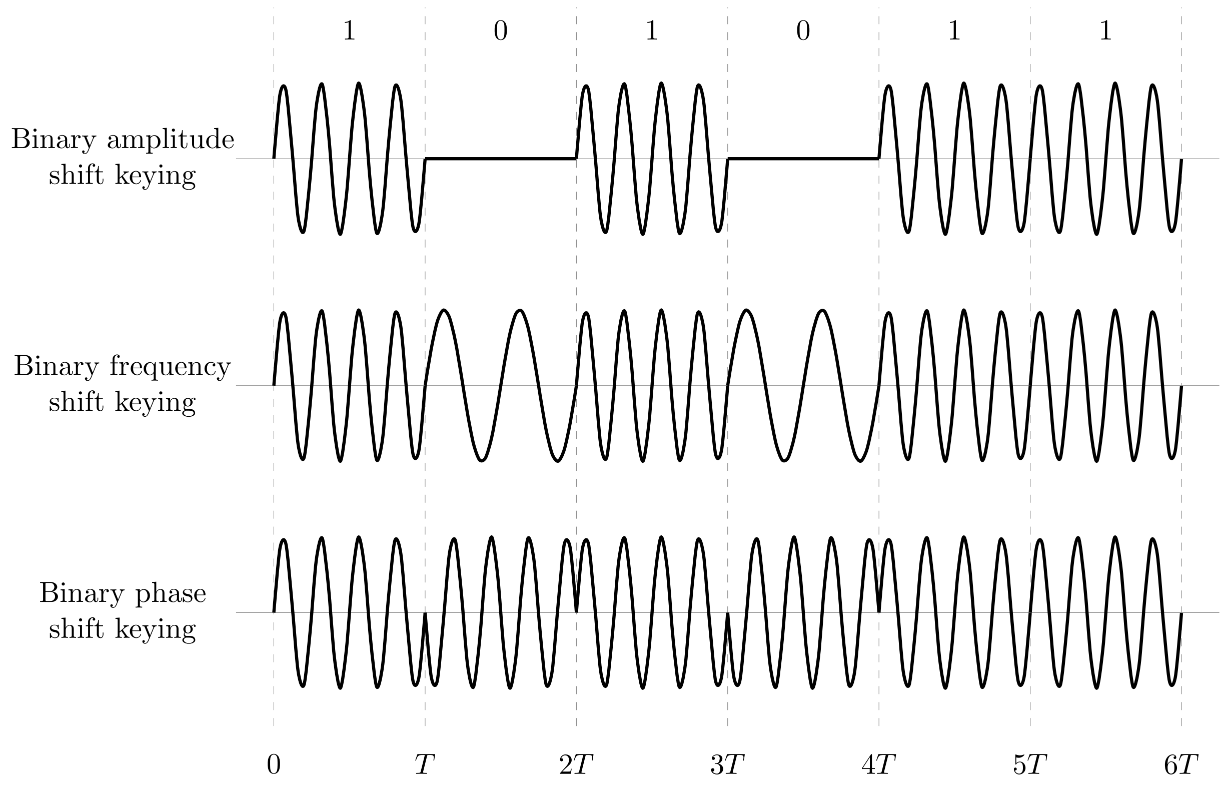

1
0
1
0
1
1
Binary amplitude shift keying
Binary frequency shift keying
Binary phase shift keying
0
T
$2T$
$3T$
$4T$
$5T$
$6T$

Figure 3.2: Illustration of binary ASK, FSK and PSK modulation schemes

3.2 Narrowband Communication

Communication through a narrowband channel involves the mapping of different signals to the different values of the amplitude, frequency or phase of a sinusoidal electromagnetic wave which is also called a *carrier*.

Amplitude Shift Keying

The simplest amplitude modulation scheme for narrowband channels is binary *amplitude shift keying (ASK)* which corresponds to mapping the bit 1 to a sinusoidal signal and the bit 0 to no signal at all. Then a string of bits $\{b_k : k = 0, 1, \ldots, N\}$ can be mapped to a sequence of pulses

$$s(t) = \sum_{k=0}^{N} p_k(t - kT) \tag{3.3}$$

where each pulse has duration T and is given by

$$p_k(t) = \begin{cases} A_c \sin(2\pi f_c t) & \text{if } b_k = 1 \\ 0 & \text{if } b_k = 0 \end{cases} \tag{3.4}$$

for $t \in [0, T]$, where A_c is the amplitude of the transmitted carrier and f_c is the carrier frequency. This is illustrated in Figure 3.2. Non-binary ASK schemes are also possible which involve the mapping of groups of bits to multiple amplitude levels of the carrier wave.

Frequency Shift Keying

A modulation scheme which modifies the frequency of the carrier wave according to the digital information to be sent is called *frequency shift keying (FSK)* scheme. For example, a binary FSK scheme maps the bit 0 to a sinusoidal signal with frequency f_1 and a bit 1 to a sinusoidal signal with frequency f_2. As before, a string of bits $\{b_k : k = 0, 1, \dots, N\}$ is mapped to sequence of pulses as in Equation (3.3) where each pulse is of duration T and is given by

$$p_k(t) = \begin{cases} A_c \sin(2\pi f_1 t) & \text{if } b_k = 0 \\ A_c \sin(2\pi f_2 t) & \text{if } b_k = 1 \end{cases} \tag{3.5}$$

for $t \in [0, T]$.

Phase Shift Keying

A modulation scheme which modifies the phase of the carrier wave according to the digital information to be sent is called *phase shift keying (PSK)* scheme. For example, a binary PSK (BPSK) scheme maps the bit 0 to a sinusoidal signal whose phase is shifted by π and a bit 1 to a sinusoidal signal with no phase shift. As before, a string of bits $\{b_k : k = 0, 1, \dots, N\}$ is mapped to sequence of pulses as in Equation (3.3) where each pulse is of duration T and is given by

$$p_k(t) = \begin{cases} A_c \sin(2\pi f_c t + \pi) & \text{if } b_k = 0 \\ A_c \sin(2\pi f_c t) & \text{if } b_k = 1 \end{cases} \tag{3.6}$$

for $t \in [0, T]$.

Quadrature Modulation

The modulation schemes just described can transmit only one bit per pulse duration T. The orthogonality of the sine and cosine waves can be exploited to increase the number of bits transmitted per pulse duration resulting in schemes called *M-ary phase shift keying* and *quadrature amplitude modulation (QAM)*. The general form of the transmitted signals under these schemes is given by

$$s(t) = A\cos(2\pi f_c t) + B\sin(2\pi f_c t) \tag{3.7}$$

where A and B take discrete values depending on the bits to be transmitted. Constraining the sum of A^2 and B^2 to unity results in M-ary PSK signals. For example, when A and B take values from the set $\left\{+\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}\right\}$, we obtain the modulation scheme known as *quadrature phase shift keying (QPSK)*.

Orthogonal Frequency-Division Modulation

Orthogonal frequency-division multiplexing (OFDM) is an example of a *multi-carrier modulation* technique. In multi-carrier modulation, the data is simultaneously modulated onto multiple sub-carriers using a modulation technique like QAM or PSK. The advantage of doing this is that the data can be recovered even if some of the sub-carriers are subjected to severe attenuation. In OFDM, the sub-carriers are chosen to be orthogonal and the data to be transmitted is divided into parallel data streams where each stream modulates one sub-carrier. For example, an OFDM signal having N sub-carriers each modulated by BPSK over a single symbol duration T is given by

$$s(t) = \sum_{k=0}^{N-1} b_k \sin\left(2\pi \left[f_c + \frac{k}{T}\right] t\right) \tag{3.8}$$

where $t \in [0, T]$ and $b_k \in \{-1, +1\}$. The sub-carrier spacing of $\frac{1}{T}$ makes them orthogonal over each symbol period.

Demodulation of Narrowband Signals

The demodulation of modulated sinusoidal signals requires knowledge of the carrier frequency, carrier phase and the symbol timing of the received signal. Even though the receiver knows the value of the transmitted carrier frequency and tunes its local oscillator to match this frequency, there might be a mismatch between the received frequency and the local oscillator's frequency. This mismatch is due to deviation of the oscillators at the transmitter and receiver from the actual frequency or due to *Doppler effect* when the transmitter is moving relative to the receiver. The carrier phase of the received signal is the sum of the random phase of transmitter's oscillator relative to the receiver's oscillator, the channel phase response and the phase due to the transmission delay. The symbol timing, i.e. the boundaries of the symbols in the received signal, is not known because of the transmission delay and the mismatch between the transmitter and receiver clocks. For example, a BPSK pulse of the form

$$s(t) = a_k p(t) \sin(2\pi f_c t), \tag{3.9}$$

where $p(t)$ is a baseband pulse which is non-zero in the interval $[0, T]$ and

$$a_k = \begin{cases} +1 & \text{if bit is 1} \\ -1 & \text{if bit is 0} \end{cases} \tag{3.10}$$

when passed through a non-dispersive channel, yields a received signal of the form

$$r(t) = A a_k p(t-\tau) \sin\left[2\pi(f_c + f_d)(t-\tau) + \theta)\right] + n(t), \tag{3.11}$$

where A is the received amplitude, f_d is the Doppler shift, θ is the received carrier phase, τ is the transmission delay and $n(t)$ is a noise process. Defining $\phi = \theta - 2\pi(f_c + f_d)\tau$, we can write the received signal as

$$r(t) = A a_k p(t-\tau) \sin\left[2\pi(f_c + f_d)t + \phi)\right] + n(t), \tag{3.12}$$

If the receiver knows f_d, ϕ and τ, the value of a_k can be recovered by correlation with $p(t-\tau)\sin\left[2\pi(f_c + f_d)t + \phi\right]$ followed by low-pass filtering. This is an example of a *coherent* demodulation scheme which is any demodulation scheme which requires knowledge of the carrier phase ϕ. There are also *noncoherent* demodulation schemes which achieve demodulation without the knowledge of the carrier phase.

The estimation of the carrier frequency is called *carrier synchronization* or *carrier recovery.* The estimation of the carrier phase is called *carrier phase synchronization* and is performed using a *phase-locked loop (PLL)* circuit. The estimation of the symbol timing is called *symbol timing synchronization* and is performed using a DLL circuit as in the baseband channel case.

In summary, all the physical layer issues of baseband channels carry over to narrowband channels with the addition of the issues of carrier phase and frequency estimation. ISI is an issue even in narrowband channels (even though we did not mention it) and requires pulse-shaping or equalization to mitigate it. In addition to channel estimation and initial timing estimation, the preamble also helps in carrier recovery and phase synchronization in narrowband channels.

3.3 Spread Spectrum Communication

The term *spread spectrum modulation* refers to modulation techniques which result in a transmitted signal bandwidth which is much wider than the minimum bandwidth required for communication. For example, a spread spectrum technique could use a signal having bandwidth 1 MHz to communicate information which can be communicated using narrowband techniques using a signal having bandwidth 5 kHz. Spread spectrum modulation techniques are used in the physical layer of many wireless commercial and military communication networks like 802.11, GPS and CDMA2000. The advantages of spread spectrum techniques include resistance to jamming and interference, low probability of interception, resistance to fading and suitability for ranging applications. But the main reason for their popularity lies in their inherent ability to provide shared channel access in multiple access channels in a decentralized manner.

The main principle behind spread spectrum's ability to enable multiple access can be explained by considering two sources (S_1 and S_2) who want to transmit a single bit each over a multiple access channel to a single destinations D. Suppose they use amplitude modulation and S_i ($i = 1, 2$) transmits the following signal

$$u_i(t) = a_i v_i(t) \tag{3.13}$$

where $a_i \in \{-1, +1\}$. The signals $v_i(t)$ have the property that they are approximately orthogonal irrespective of the relative time-shift between them, i.e. they satisfy the following property

$$\int_{-\infty}^{\infty} v_1(t) v_2(t-\tau)\, dt \approx 0 \tag{3.14}$$

for all values of τ. The received signal at the destination D is given by

$$\begin{aligned} r(t) &= A_1 u_1(t-\tau_1) + A_2 u_2(t-\tau_2) & (3.15)\\ &= A_1 a_1 v_1(t-\tau_1) + A_2 a_2 v_2(t-\tau_2) & (3.16)\end{aligned}$$

where A_i ($i = 1, 2$) and τ_i ($i = 1, 2$) represent the attenuation and transmission delay of S_i's signal, respectively. We have neglected the noise term in the received signal for convenience. If the destination now estimates τ_1 and correlates the received signal with $v_1(t-\tau_1)$, it can obtain the value of a_1 as shown below.

$$\begin{aligned} \int_{-\infty}^{\infty} r(t) v_1(t-\tau_1)\, dt &= a_1 A_1 \int_{-\infty}^{\infty} v_1^2(t-\tau_1)\, dt + a_2 A_2 \int_{-\infty}^{\infty} v_2(t-\tau_2) v_1(t-\tau_1)\, dt \\ &\approx a_1 A_1 & (3.17)\end{aligned}$$

Similarly, the value of a_2 can be obtained by estimating τ_2 and correlating the received signal with $v_2(t-\tau_2)$. Thus we are able to recover the values of the a_i's and consequently the information bit each source wants to communicate even though the transmitted signals overlap at the receiver. The approximate orthogonality property of the transmitted signals shown in Equation (3.14) is crucial for this to be possible. The signals used in spread spectrum modulation are able to provide this property at the cost of occupying a wide bandwidth.

There are three main categories of spread spectrum techniques - *direct sequence (DS)*, *frequency hop (FH)* and *time hop (TH)*. Hybrid spread spectrum techniques also exist which are a combination of these basic techniques. The main idea is to introduce high rate variations in the signal used to transmit the information resulting in a signal which occupies a larger bandwidth.

Direct Sequence Spread Spectrum

In *direct sequence spread spectrum (DSSS)*, the bandwidth occupied by a PSK signal is increased by multiplying it with high rate pseudo-random sequence of phase modulated pulses. For example, consider a BPSK signal given by

$$m(t) = \sqrt{2P} \sum_{k=-\infty}^{\infty} b_k p(t - kT) sin(2\pi f_c t), \tag{3.18}$$

where $b_k \in \{-1, +1\}$ and $p(t)$ is a pulse of duration T. We multiply the data $m(t)$ with a spreading signal given by

$$a(t) = \sum_{l=-\infty}^{\infty} a_l \psi(t - lT_c), \tag{3.19}$$

where $a_l \in \{-1, +1\}$ is a pseudo-random sequence called the *spreading sequence*, $\psi(t)$ is a pulse of duration T_c called the *chip waveform*. The value of T_c is chosen such that $T = NT_c$ for some positive integer $N > 1$. Then the spread spectrum signal is given by

$$s(t) = m(t)a(t) = \sqrt{2P} \sum_{l=-\infty}^{\infty} b_{\lfloor \frac{l}{N} \rfloor} a_l \psi(t - lT_c) \sin(2\pi f_c t) \tag{3.20}$$

where $\lfloor x \rfloor$ is the largest integer less than or equal to x. This is illustrated in Figure **??**. The bandwidth expansion can be seen by looking at the power spectrum of the spread spectrum signal which for the case when $\psi(t)$ is a rectangular pulse is given by

$$\Phi_s(f) = \frac{PT_c}{2} \left[\frac{\sin^2 [\pi(f - f_c)T_c]}{[\pi(f - f_c)T_c]^2} + \frac{\sin^2 [\pi(f + f_c)T_c]}{[\pi(f + f_c)T_c]^2} \right] \tag{3.21}$$

which is basically the sum of two frequency-shifted squared sinc functions. The main lobes of the squared sinc functions in this power spectrum are N times wider than the main lobes in the power spectrum of $m(t)$ which for the case when $p(t)$ is a rectangular pulse is given by

$$\Phi_m(f) = \frac{PT}{2} \left[\frac{\sin^2 [\pi(f - f_c)T]}{[\pi(f - f_c)T]^2} + \frac{\sin^2 [\pi(f + f_c)T]}{[\pi(f + f_c)T]^2} \right] \tag{3.22}$$

The approximately orthogonal property is obtained by using different spreading signals $a(t)$ for different users.

The main disadvantage of using DSSS signals is that the timing synchronization needs to be more accurate. It has to be correct to the order of a chip duration T_c while in narrow band signals the timing had to correct to the order of the bit duration T.

Frequency Hop Spread Spectrum

In *frequency hop spread spectrum (FHSS)*, the bandwidth of a narrowband signal is increased by pseudo-randomly choosing the carrier frequency from a discrete set of carrier frequencies. Consider a binary FSK signal given by

$$m(t) = \sqrt{2P} \sum_{k=-\infty}^{\infty} p_T(t - kT) sin\left[2\pi(f_c + f_k)t\right], \tag{3.23}$$

where $f_k \in \{f_1, f_2\}$, f_c is the carrier frequency and $p_T(t)$ is a pulse of duration T. Consider a pseudo-random sequence of carrier frequencies $\{f_c^l\}$ where each value in the sequence is picked from a discrete set of frequencies $\{f_{c,1}, f_{c,2}, \ldots, f_{c,M}\}$. The narrowband signal $m(t)$ is spread to a wider bandwidth by changing its carrier frequency every T_c seconds to the next value in the sequence $\{f_c^l\}$ which is called the *frequency hopping code*. This process is called frequency hopping and for $T = NT_c$ the resulting signal is given by

$$s(t) = \sqrt{2P} \sum_{l=-\infty}^{\infty} p_{T_c}(t - lT_c) sin \left[2\pi (f_c^l + f_{\lfloor \frac{l}{N} \rfloor})t \right]. \tag{3.24}$$

This example is in fact illustrating *fast hopping* where the hopping rate ($\frac{1}{T_c}$) is larger than the symbol rate ($\frac{1}{T}$). When the symbol rate is larger than the hopping rate, the situation is called *slow hopping*.

The main challenge in using FHSS is synchronizing the receiver with the transmitter's frequency hopping code.

Time Hop Spread Spectrum

In *time hop spread spectrum (THSS)*, the bandwidth of a narrowband signal is increased by pseudo-randomly changing the position of the pulses present in the signal. For example, consider a BPSK signal given by

$$m(t) = \sqrt{2P} \sum_{k=-\infty}^{\infty} b_k p_T(t - kT) sin(2\pi f_c t), \tag{3.25}$$

where $b_k \in \{-1, +1\}$ and $p_T(t)$ is a pulse of duration T. We multiply the data $m(t)$ with a spreading signal given by

$$a(t) = \sqrt{\frac{T}{T_c}} \sum_{l=-\infty}^{\infty} p_{T_c}(t - lT - a_l T_c), \tag{3.26}$$

where $a_l \in \{0, 1, \ldots, N-1\}$ is a pseudo-random sequence called the *time hopping code* and $p_{T_c}(t)$ is a pulse of duration T_c. As before, the value of T_c is chosen such that $T = NT_c$ for some positive integer $N > 1$. Then the spread spectrum signal is given by

$$s(t) = m(t)a(t) = \sqrt{\frac{2PT}{T_c}} \sum_{l=-\infty}^{\infty} b_l p_{T_c}(t - lT - a_l T_c) \sin(2\pi f_c t). \tag{3.27}$$

It can be shown that this signal occupies N times the bandwidth occupied by $m(t)$.

The main challenge in using THSS is synchronizing the receiver with the transmitter's time hopping code.

Chapter 4

Data Link Layer

The data link layer is concerned with providing error-free communication between adjacent nodes in a communication network. To be more specific, its goal is to transfer information bits from the network layer on a source node to the network layer on an adjacent destination node. The main functions of the data link layer are the following.

- *Framing:* The data link layer breaks up the information bit strings passed on from the network layer into smaller bit strings and encapsulates them into *frames.* This framing of the information enables the next three functions of the data link layer possible.
- *Error control:* Since the combination of the physical channel between adjacent nodes and the physical layers in the nodes behaves like an unreliable bit pipe, the data link layer is responsible for converting this into an error-free bit pipe.
- *Flow control:* The data link layer regulates the flow of data between adjacent nodes such that buffer overflows do not occur.
- *Medium access control:* When the channel between adjacent nodes is a multiple access channel, a sublayer of the data link layer called the MAC layer coordinates the channel access.

4.1 Framing

The network layer at a node passes information bit strings called *packets* to the data link layer to transfer to the adajcent node along the way to the final destination node. The data link layer breaks every packet into smaller bit strings and encapsulates them into frames. If the packets are small enough, breaking of the packets may not be necessary and encapsulation may be sufficient. Each frame consists of a *header* preceding the information bits and a *trailer* following the information bits. The information bit string between the header and trailer is sometimes referred to as the *payload* but this terminology is not restricted to the data link layer. The whole frame is then passed on to the physical layer for transmission over the physical channel. This is illustrated in Figure 4.1. The framing operation allows the data link layer to provide the error control, flow control and medium access control capabilities. For instance, the CRC bits which are used for error detection are typically stored in the trailer appended by the data link

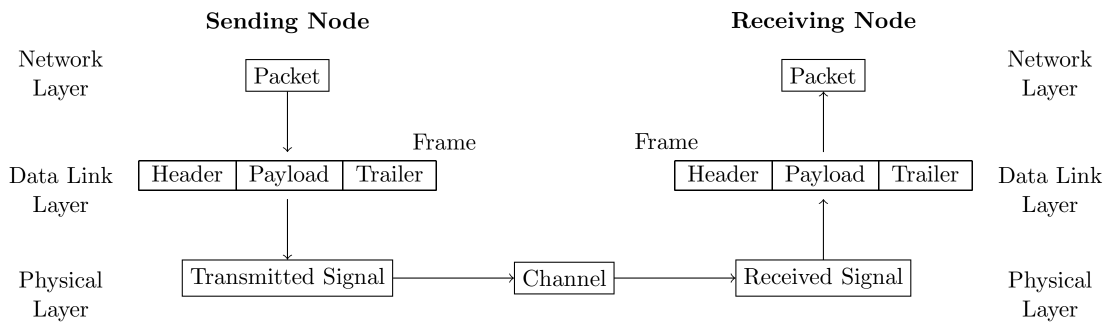
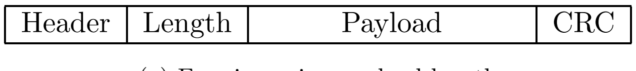


Figure 4.1: Illustration of framing at the data link layer

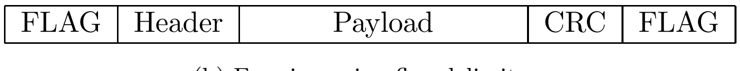
layer. The header contains the destination MAC address which is used to identify the intended receiver of the packet when the channel is a multiple access channel.

The data link layer on the receiving node is provided a sequence of bits from the physical layer. Determining where the frames begin and end is the main challenge faced by the receiving data link layer. It needs to identify the header and trailer before it can do subsequent processing. Identifying the header is easy since it appears in the beginning of a frame. Identifying the trailer is more challenging because the length of the payload may be variable. One way to solve this problem is to include the length of the payload in the frame header. This is illustrated in Figure 4.2a. The problem with this approach is that an error in the length field can cause some other bits to be interpreted as the CRC bits. For example, suppose the original length was 1000 and a channel error caused the length to be interpreted as 500. Then some bits in the middle of the data portion will be taken to be the CRC bits resulting in an error detection being declared by the receiver with high probability. Such a length error can not only affect the current frame but also the subsequent frames even if they are error-free.

An alternative framing strategy involves delimiting the frame with special bytes called *flag bytes*. This is illustrated in Figure 4.2b. The data link layer on the receiving node parses the bit sequence provided by the physical layer for the flag bytes and takes the bits between two consecutive flag bytes to be the frame. Of course, if the flag bytes are corrupted by channel errors the frame cannot be correctly identified. Even if there are no channel errors the appearance of the flag byte in the payload can cause the frame boundaries to be erroneously identified. The solution to this problem is to insert a special escape byte (ESC) before the flag byte in the payload to distinguish it from a flag byte at the frame boundary. This is called *byte stuffing*. Whenever the receiver sees an escape byte followed by the flag byte it will infer that the flag byte appeared in the payload and not at the end of a frame. It will then simply remove or *destuff* the escape byte before passing the payload bits to the network layer. This scheme will work as it is if the escape byte is inserted only before a flag byte. An escape byte which is not followed by a



(a) Framing using payload length



(b) Framing using flag delimiters

Figure 4.2: Illustration of different framing strategies

Payload bytes	After byte stuffing	After byte destuffing
0xAB FLAG 0xCD	0xAB ESC FLAG 0xCD	0xAB FLAG 0xCD
0xAB ESC 0xCD	0xAB ESC ESC 0xCD	0xAB ESC 0xCD
0xAB ESC FLAG 0xCD	0xAB ESC ESC ESC FLAG 0xCD	0xAB ESC FLAG 0xCD

Table 4.1: Illustration of byte stuffing and destuffing

flag byte is not destuffed. However, in most data link layer protocols, the escape byte is inserted before other control bytes as well. Then the receiver has to check if the escape byte is followed by any one of the control bytes before destuffing it. This can be a time-consuming operation if the number of possible control bytes is large. A simpler strategy is to insert another escape byte before the appearance of every escape byte in the payload at the sending data link layer. At the receiving data link layer, every pair of escape bytes is destuffed to a single escape byte and an escape byte followed by a flag byte is destuffed to a flag byte. This is illustrated in Table 4.1 where FLAG denotes the flag byte, ESC denotes the escape byte and the other payload bytes are given in hexadecimal notation.

Another approach to framing involves *bit stuffing* instead of byte stuffing. As before the start and end of a frame is indicated by a flag byte which in this case is chosen to be the byte 01111110. To deal with the issue of the flag byte appearing in the payload, the sending data link layer inserts a 0 bit whenever it encounters five consecutive 1s in the payload. This is illustrated in Figure 4.3.

In reality, many data link layer protocols use the payload length and the flag delimiters together to achieve framing. This has the advantage of more robustness against channel errors. The payload length is used to calculate the end of the frame and the presence of a flag at that position is checked. If the the expected flag is absent either due to channel errors corrupting the length field or the flag itself, the frame is discarded as erroneous.

4.2 Error Control

The most crucial function of the data link layer is to convert the unreliable bit pipe provided by the physical layer into a error-free bit pipe. This is achieved using the combination of an error-detecting code like cyclic redundancy check (CRC) and automatic repeat request (ARQ). Although CRC is just one type of error-detecting code, it is the most prevalent one in usage today and hence it has become synonymous with error detection in networking literature. ARQ uses a combination of timeouts and acknowledgements to ensure frame retransmissions when frame errors are detected by the CRC.

In the introductory chapter, we mentioned forward error correction (FEC) as another method

Payload bits 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0

After bit stuffing 1 0 1 0 1 1 1 1 1 $\underline{0}$ 1 1 1 1 1 $\underline{0}$ 1 1 1 1 1 $\underline{0}$ 1 0 1 1 0

After destuffing 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0

Figure 4.3: Illustration of bit stuffing where the underlined bits are introduced by the bit stuffing operation

for enabling error-free communication which uses redundancy to correct errors. We observed that CRC was essential for ARQ to work correctly but FEC was not. We also looked at the repetition code as the simplest FEC scheme. In this course, we will not go into the details of more complicated FEC schemes as that will require a lot more time than what can be afforded for such a topic which can span a whole course on its own. A final remark is due, however, regarding which layer FEC schemes belong to. Although they should naturally belong to the data link layer since they help provide error-free communication between adjacent nodes, they are usually implemented as part of the physical layer. For instance, most physical layer specifications of current wireless standards like 802.11 and WiMAX include the description of the particular FEC scheme which can be used. One reason for this is that the physical layer algorithms are implemented in hardware.The FEC encoding/decoding operations can be computationally intensive and can be performed faster in hardware. Another reason is that soft-decision decoding algorithms for FEC schemes which take into account the received signal values have better performance than hard-decision decoding algorithms. The received signal values are available only at the physical layer.

4.2.1 Cyclic Redundancy Check

The basic principle behind an error detection code is to append a fixed number of *check bits* to the payload at the sender which are a deterministic function of the payload bits. At the receiver, the check bits are recalculated using the same deterministic function acting on the received payload bits and compared to the received check bits. The deterministic function is chosen such that errors in the decoder output will cause a discrepancy between the recalculated check bits and the decoded check bits. When a discrepancy arises, an error is detected. The check bits are also called the *checksum* of the payload. This terminology is a result of the fact that the earliest error detection codes just performed a binary summation of the bytes in the payload to generate the check bits.

The number of check bits is much less than the number of payload bits and hence multiple payload bit strings will map to the same checksum. So there is a possibility of channel errors going undetected when they transform the actual payload bits to another payload bit string which yields the same check bits. To alleviate this problem, the error detection code is chosen such that a number of errors less than a fixed value cannot result in a valid payload-checksum pair.

Binary Polynomials

The study of CRC is greatly simplified if we represent bit strings using binary polynomials, i.e. polynomials with coefficients which are either 0 or 1. For example, the bit string 101 is mapped to X^2+1. In general, the one-to-one correspondence between n-length bit strings to polynomials of maximum degree $n-1$ is given by the following.

$$b_{n-1}b_{n-2}\cdots b_2b_1b_0 \longleftrightarrow b_{n-1}X^{n-1}+b_{n-2}X^{n-2}+\cdots+b_2X^2+b_1X+b_0 \tag{4.1}$$

Here are some more examples for the case $n=8$.

$$\begin{array}{lcl}
00000000 & \longleftrightarrow & 0 \\
00001000 & \longleftrightarrow & X^3 \\
10101010 & \longleftrightarrow & X^7+X^5+X^3+X \\
11111111 & \longleftrightarrow & X^7+X^6+X^5+X^4+X^3+X^2+X+1
\end{array}$$

If $b_{n-1} = 1$, the degree of the corresponding binary polynomial is $n-1$ and the degree is less than $n-1$ if $b_{n-1} = 0$. We will need to perform the arithmetic operations of addition, subtraction, multiplication and division on these polynomials.

Addition of binary polynomials is similar to addition of polynomials with real or complex coefficients except that the coefficients obey the rules of binary arithmetic, i.e. $0+0=0-0=0$, $1+1=1-1=0$, $1+0=0+1=1-0=1$, $0-1=1$. For example,

$$\left(X^3+X+1\right)+\left(X^2+X+1\right)=X^3+X^2+(1+1)X+1+1=X^3+X^2.$$

Also,

$$\left(X^3+X+1\right)+\left(X^3+X+1\right)=(1+1)X^3+(1+1)X+1+1=0.$$

The addition of a binary polynomial with itself always gives zero. Thus, a binary polynomial is its own additive inverse, i.e. $b(X)=-b(X)$ for all binary polynomials $b(X)$. This also means that addition of two binary polynomials is the same as subtracting the one from the other, i.e.

$$a(X)+b(X)=a(X)-b(X)=-a(X)+b(X)=-a(X)-b(X),$$

for all binary polynomials $a(X)$ and $b(X)$.

Similarly, multiplication of binary polynomials is similar to the multiplication of polynomials with real or complex coefficients except that the coefficients obey the rules of binary arithmetic. For example,

$$\begin{aligned}(X+1)\left(X^3+X+1\right) &= X\left(X^3+X+1\right)+\left(X^3+X+1\right)\\ &= X^4+X^2+X+X^3+X+1=X^4+X^3+X^2+1.\end{aligned}$$

Note that multiplying a polynomial by X shifts the corresponding bit string one position to the left. For example, when $n=8$

$$\begin{aligned}X^3+X+1 &\longleftrightarrow 00001011\\ X\left(X^3+X+1\right)=X^4+X^2+X &\longleftrightarrow 00010110.\end{aligned}$$

One interesting fact is that the square of a binary polynomial is equal to the sum of the squares of the individual terms.

$$\begin{aligned}\left(X^3+X+1\right)^2 &= X^6+X^2+1+X^3\cdot X+X\cdot X^3+X^3\cdot 1+1\cdot X^3+X\cdot 1+1\cdot X\\ &= X^6+X^2+1.\end{aligned}$$

This property actually holds for any 2^i-power of a polynomial.

Division of binary polynomials is also similarly related to the division of ploynomials with real coefficients. An example is as follows.

$$\begin{array}{r|l} & \quad\quad\quad\quad X^3 \quad\quad - 1 \\ \hline X^5+X^2+1) & X^8 \\ & -X^8-X^5-X^3 \\ \cline{2-2} & \quad -X^5-X^3 \\ & \quad\quad X^5 \quad\quad +X^2+1 \\ \cline{2-2} & \quad\quad -X^3+X^2+1 \end{array} \tag{4.2}$$

Of course, a -1 is equivalent to a 1.

CRC Error Detection

Suppose we have an n-bit information bit string $(m_{n-1}, m_{n-2}, \ldots, m_2, m_1, m_0)$. The corresponding binary polynomial is

$$m(X) = m_{n-1}X^{n-1} + m_{n-2}X^{n-2} + \cdots + m_1 X + m_0.$$

Suppose we append the check bit string $(c_{k-1}, c_{k-2}, \ldots, c_2, c_1, c_0)$ to the information bit string. Let the polynomial corresponding to the check bits be

$$c(X) = c_{k-1}X^{k-1} + c_{k-2}X^{k-2} + \cdots + c_1 X + c_0.$$

Then the transmitted bit string will be $(m_{n-1}, m_{n-1}, \ldots, m_1, m_0, c_{k-1}, c_{k-2}, \ldots, c_1, c_0)$ which corresponds to the binary polynomial

$$s(X) = m(X)X^k + c(X). \tag{4.3}$$

Remember that multiplication of a polynomial by X^k will cause the corresponding bit string to shift to the left by k positions. Thus Equation (4.3) corresponds to the shifting the message bits to the left to make room for the check bits and inserting the check bits in the vacated positions.

When CRC is used to generate the check bits, $c(X)$ is called the *CRC polynomial.* The CRC polynomial $c(X)$ is a function of the information polynomial $m(X)$, defined in terms of a *generator polynomial* of degree k,

$$g(X) = X^k + g_{k-1}X^{k-1} + g_{k-2}X^{k-2} + \cdots + g_2 X^2 + g_1 X + 1 \tag{4.4}$$

as follows

$$c(X) = \text{Remainder}\left[\frac{m(X)X^k}{g(X)}\right].$$

So the CRC polynomial is the remainder from dividing the left-shifted information polynomial $m(X)X^k$ by the generator polynomial $g(X)$. Notice that we have defined the generator polynomial such that the terms with degree k and 0 (the first and the last terms) cannot have a non-zero coefficient. Let $q(X)$ be the quotient resulting from dividing $m(X)X^k$ by $g(X)$. Then we have

$$\begin{aligned} m(X)X^k &= q(X)g(X) + c(X) \\ \Rightarrow m(X)X^k + c(X) &= q(X)g(X) \end{aligned}$$

But the left-hand side of the second equation is just $s(X)$ from Equation (4.3) which is the binary polynomial corresponding to the transmitted bit string. This is true for any $m(X)$ and so all the transmitted bit strings are multiples of the generator polynomial (here we are using the bit string to mean the corresponding polynomial for brevity). Also, every transmitted bit string is divisible by the generator polynomial. So if there are no errors introduced by the channel, the received bit string must also be divisible by the generator polynomial. Thus, if $r(X)$ is the received bit string the error detection procedure is the following.

$$\text{If Remainder}\left[\frac{r(X)}{g(X)}\right] \neq 0, \text{declare error detection.} \tag{4.5}$$

If $s(X)$ and $r(X)$ represent the transmitted and received bit strings respectively. Then $r(X) = s(X) + e(X)$ where $e(X)$ represents the bit string which has 1s in the error locations, i.e. the positions where $s(X)$ and $r(X)$ differ. For example, let 1001 be the transmitted bit string represented by $s(X) = X^3 + 1$ and 1011 be the received bit string represented by $r(X) = X^3 + X + 1$. Then $e(X) = X$ which represents the error bit string 0010.

$$\text{Remainder}\left[\frac{r(X)}{g(X)}\right] = \text{Remainder}\left[\frac{s(X)+e(X)}{g(X)}\right] = \text{Remainder}\left[\frac{e(X)}{g(X)}\right] \tag{4.6}$$

From Equations (4.5) and (4.6), we see that an error will be detected if $e(X)$ is not divisible by $g(X)$. If there are no errors introduced by the channel, $e(X) = 0$ and the remainder after division by $g(X)$ will be zero. If $e(X) \neq 0$, errors will go undetected only if the remainder is zero. This happens only if

$$e(X) = g(X)z(X) \tag{4.7}$$

for some non-zero polynomial $z(X)$.

Suppose a single bit error occurs. Then $e(X) = X^l$ where $0 \leq l \leq \deg[s(X)]$. Since we chose the generator polynomial such that the highest and lowest order terms are non-zero (X^k and 1, see Equation (4.4)), $g(X)z(X)$ must also have at least two non-zero terms. Let $z(X)$ be a non-zero polynomial with highest and lowest order terms of degree i and j, respectively. Then $z(X) = X^i + z_{i-1}X^{i-1} + \cdots + X^j$. If $z(X)$ has only one term, then $i = j$. Now $g(X)z(X)$ will have at least the two non-zero terms X^{k+i} and X^j and hence

$$X^l \neq g(X)z(X)$$

for any $z(X)$. Thus the remainder of dividing X^l by $g(X)$ will never be zero and hence *all single errors are detected*.

Suppose a double error occurs. Then $e(X) = X^i + X^j = X^j(X^{i-j} + 1)$ where $i > j$. Now X^j is not divisible by $g(X)$ by our previous argument about single errors. So for $e(X)$ to be a multiple of $g(X)$, $X^{i-j} + 1$ should be a multiple of $g(X)$. In other words, $g(X)$ should divide $X^{i-j} + 1$ for $e(X)$ to be an undetectable error pattern. If $g(X)$ is chosen to be a *primitive polynomial* of degree k, then the smallest value of m for which $g(X)$ divides $X^m + 1$ is $2^k - 1$. So if $g(X)$ is chosen to be a primitive polynomial of degree k and the transmitted bit string length is chosen to be at most $2^k - 1$, then $X^{i-j} + 1$ cannot be divisible by $g(X)$. This is because if the transmitted bit string is of length at most $2^k - 1$, then $0 \leq i, j \leq 2^k - 2$ and hence $i - j \leq 2^k - 2$. This proves that *all double errors are detected*.

Let us now consider the detection of *burst errors*. Any possible error pattern can be characterized as a burst error. The *burst length* of a burst error is defined as the number of positions from the first error to the last error including the endpoints. For example, the error pattern 0010010 has burst length 4. In the corresponding error polynomial $e(X) = X^4 + X$, the highest and lowest order terms differ in degree by 3. Since the highest and lowest order terms in $g(X)$ differ in degree by k, by Equation (4.7) the highest and lowest order terms in any undetectable error polynomial $e(X)$ also differ by at least k. Hence the burst length of an undetectable error pattern is at least $k + 1$. The lower bound of $k + 1$ on an undetectable error burst length implies that *all error patterns with burst length less than $k + 1$ will be detected*.

Suppose a burst error of length $k + 1$ occurs. It will be of the form $e(X) = X^i(X^k + e_{k-1}X^{k-1} + \cdots + e_2X^2 + e_1X + 1)$, where $0 \leq i \leq n - 1$. For each i, there are 2^{k-1} possible error patterns. Of these, only one is undetectable, namely $X^i g(X)$. Thus the fraction of undetectable error patterns of burst length $k + 1$ is $2^{-(k-1)}$.

If the burst length l is greater than $k + 1$, there are 2^{l-2} such error patterns starting at position l and ending at position $i + l - 1$. Of these, the undetectable error patterns will be of the form

$X^i a(X) g(X)$ where $a(X) = X^{l-k-1} + a_{l-k-2}X^{l-k-2} + \cdots + a_1 X + 1$. For each i, there are 2^{l-k-2} such undetectable error patterns. Thus the fraction of undetectable error patterns of length $l > k + 1$ is 2^{-k}.

In practice, the generator polynomial is chosen to be the product of a primitive polynomial of degree $k - 1$ and the polynomial $X + 1$. A polynomial $e(X)$ is divisible by $X + 1$ if and only if it contains an even number of non-zero coefficients. This ensures that all error patterns with odd number of errors in the transmitted bit string are detected. Additionally, the primitive polynomial detects all single errors and all double errors as long as the transmitted bit string length is less than 2^{k-1}. So the CRC corresponding to this generator polynomial can detect upto 3 errors in any arbitrary locations. It can also detect all burst errors of length at most k and many of the burst errors of length greater than k.

For the CRC-16 code given by the generator polynomial $g(X) = X^{16} + X^{15} + X^2 + 1 = (X + 1)(X^{15} + X + 1)$, the smallest integer m for which $g(X)$ divides $X^m + 1$ is $2^{15} - 1 = 32,767$. So if the transmitted bit string length is at most 32,767, the CRC-16 polynomial can detect all single, double, triple and odd numbers of errors. It can also detect all burst errors of length 16 or less, 99.997% of 17-bit error bursts, and 99.998% of 18-bit or longer error bursts. The CRC-32 code given by the generator polynomial

$$g(X) = X^{32}+X^{26}+X^{23}+X^{22}+X^{16}+X^{12}+X^{11}+X^{10}+X^{8}+X^{7}+X^{5}+X^{4}+X^{2}+X+1 \quad (4.8)$$

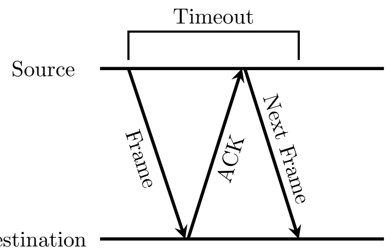
is used in Ethernet for its superior burst-error detecting capability.

4.2.2 Automatic Repeat Request

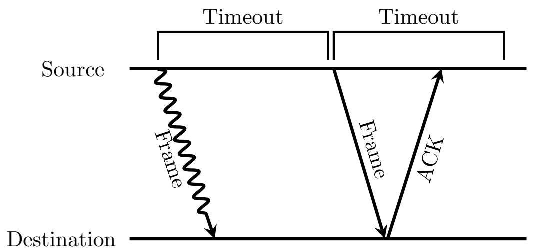
The basic principle behind automatic repeat request (ARQ) is to initiate frame retransmissions when errors are detected in the received frame. The CRC scheme discussed in the previous section is used to detect errors. Retransmissions are initiated using a combination of timeouts and acknowledgements. There are three basic types of ARQ protocols: *stop-and-wait ARQ*, *go-back-N ARQ* and *selective repeat ARQ*.

Stop-and-wait ARQ

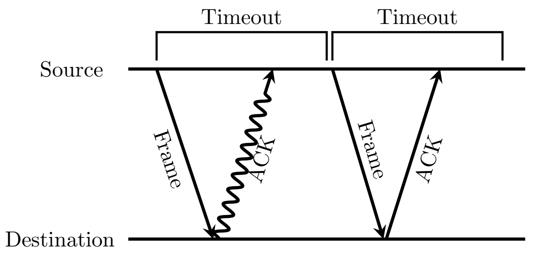
Stop-and-wait ARQ (SW ARQ) is the simplest ARQ protocol. The basic idea is to ensure that a frame has been received correctly before initiating the transmission of the next frame. When a source node sends a frame, it starts a timer and waits for an acknowledgement (ACK) from the destination.If the destination node receives a frame in which no errors are detected, it sends an ACK frame back to the sender informing it of error-free frame reception. Once an ACK for a frame arrives at the source, the next frame is transmitted. This is illustrated in Figure 4.4a. If errors are detected in the frame, then the destination does nothing. If the timer at the source exceeds a predetermined threshold, a timeout event is declared and the frame is resent. This is illustrated in Figure 4.4b. Thus the frame is retransmitted until it is received correctly at the destination. Note that the ACK frame can also be affected by errors and hence is protected by a CRC scheme. If errors are detected in the ACK frame, it is ignored. Thus frames sent from the source to the destination can be retransmitted even if they are received correctly at the destination because the corresponding ACK frame was corrupted. This is illustrated in Figure 4.4c. Frame retransmission can occur even in the case of an error-free channel due to early timeouts. If the timeout period is chosen to be too small, the timeout event occurs



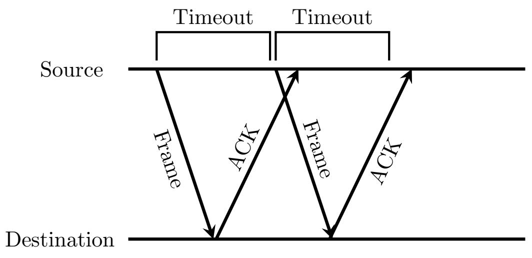
(a) The ideal case: ACK is received before timeout occurs



(b) Error case: Frame errors are detected



(c) Error case: ACK errors are detected



(d) Error case: Two copies of frame are accepted due to early timeout

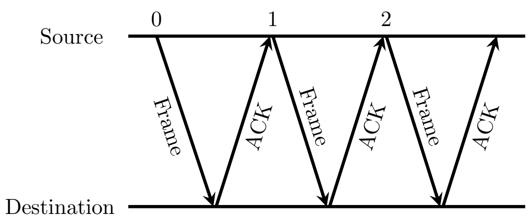
Figure 4.4: Timing diagrams illustrating stop-and-wait ARQ operation

before the ACK can reach the source and a frame retransmission occurs. This is illustrated in Figure 4.4d. To avoid unnecessary frame retransmissions, the timeout period should be chosen such that it is at least as large as a *round-trip time (RTT)*. The RTT is defined as the time taken to transmit a frame and receive an acknowledgement when no errors are introduced by the channel, i.e. the case illustrated in Figure 4.4a. The round-trip time is given by

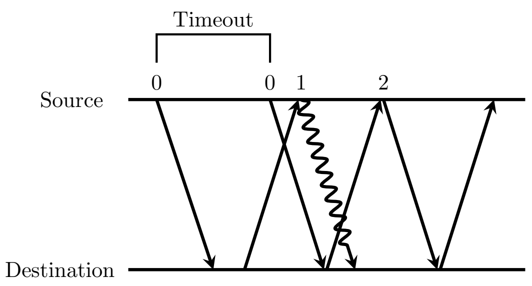
$$\text{Round-trip time} = \text{Propagation delays} + \text{Transmit duration} + \text{Processing delays} \tag{4.9}$$

The propagation delay in direction, say from source to destination, is given by $\tau = \frac{d}{c}$ where d is the distance between the nodes and c is the speed of light in the medium connecting them. So the round-trip propagation delay is 2τ. The transmit duration of the frame is given by $T_f = \frac{L_f}{D}$ where L_f is the length of the frame in bits and D is the data rate in bits per second. The transmit duration of the ACK is given by $T_a = \frac{L_a}{D}$ where L_a is the length of the ACK in bits. The total transmit duration is then given by $T_f + T_a$. The processing delay T_p is the delay between the reception of the frame at the destination and the start of the ACK transmission. The timeout should be chosen to be at least as large as $T_f + T_a + T_p + 2\tau$ to avoid unnecessary retransmissions. However, the processing delay typically cannot be predicted in advance since it may include the queueing delay for the frame, i.e. the frame may be received and be kept waiting in the receive queue before it is processed.

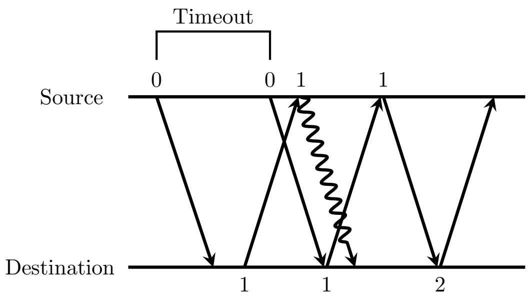
In the cases illustrated in Figures 4.4c and 4.4d, the data link layer at the destination accepts two copies of a frame and passes it along to the network layer since both copies are error-free. Comparing consecutive frames to detect duplicates is not a good strategy because the source may sometimes send some information which can result in consecutive frames having the same information. The simplest solution is to include a *sequence number* for the frame in the frame header which can be used to detect and reject duplicate frames. This is illustrated in Figure 4.5a where the numbers at the beginning of a frame transmission indicates the sequence number of the frame. Even if sequence numbers are used in the frames, delayed ACKs and corrupted frames can cause the wrong frames to be acknowledged. This is illustrated in Figure 4.5b where



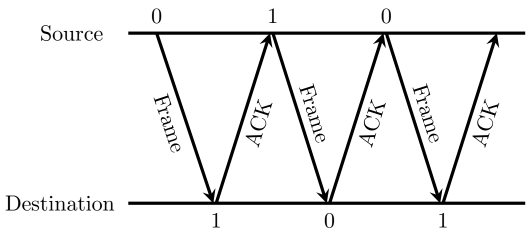
(a) Error-free case with increasing sequence numbers



(b) Error case: No sequence numbers in the ACKs



(c) Sequence numbers present in the ACKs



(d) One-bit sequence numbers

Figure 4.5: Sequence numbers to distinguish duplicate frames

a delayed ACK for the frame 0 has been incorrectly accepted as an ACK for the retransmission of frame 0 and the ACK for the retransmission has been accepted as the ACK for frame 1. Even if frame 1 is corrupted by errors, frame 2 will be transmitted by the source. The solution is to return the sequence number of the next frame expected by the destination in the ACK for a correctly received frame. This is illustrated in Figure 4.5c where the numbers at beginning of an ACK transmission indicate the sequence number of the next frame expected by the destination. Of course, the destination could include the sequence number of the currently acknowledged frame in the ACK but this is not done to conform with the other ARQ protocols where sending the sequence number of the next expected frame has a definite advantage.

Sequence numbers are embedded in the frame header and hence it is desirable to limit the range of values they can take so as to minimize the number of bits which need to be reserved for them in the header. At any moment, the ambiguity is between frame m and frame $m + 1$. At the destination, a copy of frame m is received due to an early timeout or due to the corruption of the ACK for frame m. Similarly, the source needs to distinguish between the ACKs for frames m and $m + 1$. Hence a one-bit sequence number (0 or 1) is sufficient. The sequence number alternates between 0 and 1 for successive frames. This is illustrated in Figure 4.5d.

On half-duplex links, one cannot do better than SW ARQ. However, on full-duplex links, SW ARQ can result in poor link utilization because it allows the source to have only one outstanding frame at a time. For example, consider a full-duplex link with data rate 500 kbps with a propagation delay of 25 ms. Suppose the source is sending frames of length 1000 bits and ACKs have length 100 bits. If the source starts sending a frame at $t = 0$s, the frame has been completely transmitted by the source at $t = 2$ ms and the frame has been received completely at the destination at $t = 27$ ms. Let us assume that there are no errors in the frame and the processing delay is negligible. At $t = 27.2$ ms, the ACK has completely transmitted by the destination and at $t = 52.2$ ms the ACK has been completely received by the source. So the

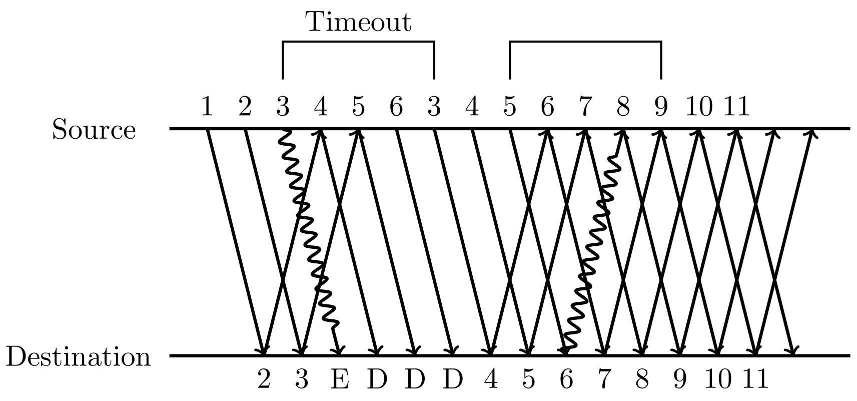


Figure 4.6: Illustration of go-back-N ARQ with $N = 4$

link from the source to the destination was idle for a fraction equal to $\frac{50.2}{52.2}$ or 96% of the time. A solution is to send N frames before waiting for ACKs. In the previous example, N should be at least 27. Once the 27th frame has been transmitted by the source t will be 54 s and the ACK for the first frame would have arrived in the error-free case. To deal with the case when frame errors occur, the value of N should be chosen such that $NT_f > T_t$ where T_f is the frame transmission time and T_t is the timeout duration. Then by the time N frame transmissions have completed, ACKs for the first few frames have arrived or a timeout for the first frame has occurred. In the former case, the source can start sending frame $N + 1$ and in the latter case it can start retransmission of frame 1.

Go-back-N ARQ

In go-back-N ARQ (GBN ARQ), the source sends N frames and then checks if the ACKs have arrived for the first few frames it sent. If the ACKs have arrived for frames 1 through k $(1 \leq k \leq N)$, it sends frames $N+1$ through $N+k$. Thus the number of unacknowledged frames at any time is N. The destination accepts frames as long as they arrive without errors. If the destination receives an erroneous frame (say frame i), it begins to reject or drop all subsequent out-of-sequence (sequence number $> i$) error-free frames without even sending an ACK until a error-free copy of frame i arrives. But if the out-of-sequence frame has sequence number less than the sequence number of the next expected frame (i.e. sequence number $< i$), the destination drops it because an error-free copy of this frame has already been accepted. However, it sends an ACK to the source because the retransmission may be due to corrupted ACKs. If the ACK for a particular frame i did not arrive at the source, it resends frames i through $i + N - 1$. Thus it resends N frames starting from the frame which was corrupted because the subsequent frames are being rejected at the destination. This is the reason for the nomenclature go-back-N. The rationale behind this apparent wastage is that the rejection of out-of-sequence error-free frames reduces the memory requirement at the destination. The data link layer at the destination needs to pass the received frames to the network layer at the destination in the order they were sent. If the destination did not reject the out-of-sequence error-free frames it would have to store them in memory while waiting for the missing frame.

The typical operation of GBN ARQ is illustrated in Figure 4.6 for the case of $N = 4$. The source is sending frames starting with sequence number 1. The destination responds to an error-free in-sequence frame with the sequence number of the next expected frame. For instance, the first frame is received without any errors and the destination responds with an ACK which has

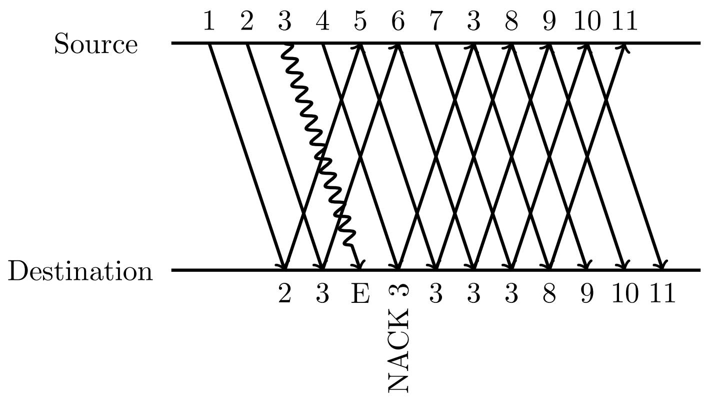


Figure 4.7: Illustration of selective repeat ARQ

sequence number 2 which corresponds to the next expected frame. Frame 3 is received in error at the destination which is denoted by the squiggly line and the letter E below the point where the frame is received. The subsequent out-of-sequence error-free frames 4, 5 and 6 are dropped by the destination which is denoted by the letter D below the point where the frame is received. The timeout for frame 3 occurs at the source and it resends frames 3, 4, 5 and 6.

The figure also illustrates the advantage of embedding the sequence number of the next expected frame in the ACK rather than the sequence number of the current received frame. Suppose the ACK correponding to frame 5 is corrupted but the ACK corresponding to frame 6 is received without errors at the source. Now the latter ACK informs the source that the sequence number of the next expected frame is 7 which implies that frame 5 was received without errors. Also this ACK arrives at the source before the timer for frame 5 expires. Thus an ACK corruption does not cause a retransmission as long as subsequent ACKs arrive without error and carry the sequence number of the next expected frame.

Selective Repeat ARQ

In selective repeat ARQ (SR ARQ), the source keeps sending frames without waiting for ACKs but restricts the number of unacknowledged frames to N. It also responds to *negative acknowledgements (NACKs)* in between frame transmissions. When the destination receives an out-of-sequence error-free frame after a corrupted frame, it sends a NACK with the sequence number of the corrupted frame embedded in it since it is the next expected frame. Upon receiving a NACK, the source retransmits the frame whose sequence number is present in it. The destinations stores the out-of-sequence error-free frames which arrive subsequent to a frame error and waits for the missing frame. While a NACK is sent in response to the first out-of-sequence error-free frame after a corrupted frame, an ACK is sent for each out-of-sequence error-free frame which does not immediately follow a corrupted frame. It contains the sequence number of the earliest corrupted frame as the next expected frame. This is illustrated in Figure 4.7.

Sliding Window Protocols and Sequence Numbers

The three ARQ protocols just described can be characterized as *sliding window* protocols. In sliding window protocols, the frames to be transmitted are numbered in an increasing sequence. The source and the destination maintain a window of frames which they can transmit and receive respectively. The source will not transmit frames which lie outside the sender window and the destination will not accept frames which lie outside the receiver window.

In SW ARQ, both the sender window and the receiver window have size one and initially contain the first frame. When the destination receives the first frame without errors, it advances its window to contain the second frame. It will then acknowledge but drop retransmitted copies of the first frame. When the source receives an acknowledgement for the first frame, it will advance its window to contain the second frame. In SW ARQ, a one-bit sequence number suffices to synchronize the window advances between the sender and the receiver.

In GBN ARQ, the sender window has size N (> 1) while the receiver window has size 1. The destination is ready to accept only the next expected frame. When an ACK for the oldest unacknowledged frame arrives at the source, it advances the sender window by one. When the source infers that frames have been received erroneously, it retransmits all the frames in its window starting with the oldest unacknowledged frame. Since the sender window size is N, the source resends N frames. As in SW ARQ, the destination advances its window by one if it receives the next expected frame without errors. In GBN ARQ, the range of sequence numbers required at the source is strictly greater than the window size N. So if m bits are used to store the sequence number in the frame header or ACK header, the number of possible sequence numbers is 2^m and the window size can be at most $2^m - 1$. The reason for this can be seen using an example. Consider the case of $m = 3$, the number of possible sequence numbers is 8. Let the sequence numbers be $\{0, 1, 2, 3, 4, 5, 6, 7\}$. Suppose the window size is $N = 2^3 = 8$. Initially, the source sends frames with sequence numbers $0, 1, \dots, 7$. Suppose all the frames are received without errors at the destination. For each error-free frame the destination increases the next expected sequence number by one. Initially it is looking for frame 0. After frame 0 is received without errors, it is looking for frame 1 and so on. Hence after frames $0, 1, \dots, 7$ are received without errors, the next expected frame (the ninth frame) has sequence number 0 again because after 7 the sequence numbers wrap around. Now suppose that the ACKs for all the frames $0, 1, \dots, 7$ are received with errors at the source. Then they are all ignored and a timeout occurs at the source resulting in a retransmission of frames 0 through 7. If frame 0 is received correctly at the destination, then it will accepted. However, the destination thinks this is the ninth frame which also has sequence number 0 and fails to recognize that it is copy of the first frame. This results in duplicate frames being passed onto the network layer. This situation can be avoided if the window size is restricted to a maximum value of $N = 2^3 - 1 = 7$. Then in the example just discussed frames $0, 1, \dots, 6$ are sent initially and the next expected frame has sequence number 7. So retransmissions of frames $0, 1, \dots, 6$ due to lost ACKS will be rejected by the destination.

In SR ARQ, both the source and destination have window size $N > 1$. The destination accepts frames which are out-of-sequence as long as they belong to the receiver window. It waits for the error-free arrival of missing frames before passing them upto the network layer. As in GBN ARQ, the source advances the sender window by one everytime it receives an ACK for the oldest unacknowledged frame. The destination advances the receiver window by one everytime it receives a frame which is the next expected frame. Since SR ARQ can accept frames out of order, the window size can be at most half the size of the range of sequence numbers. So if m bits are used to store the sequence number in the frame header or ACK header, the number of possible sequence numbers is 2^m and the window size can be at most 2^{m-1}. The reason for this can be seen using an example. Consider the case of $m = 3$, the number of possible sequence numbers is 8. Let the sequence numbers be $\{0, 1, 2, 3, 4, 5, 6, 7\}$. Suppose the window size is more than $2^{3-1} = 4$. Let$N = 5$. Initially, the source sends frames with sequence numbers $0, 1, 2, 3, 4$. Suppose all the frames are received without errors at the destination. For each error-free frame the destination advances the receive window by one. Initially it is looking for frames $\{0, 1, 2, 3, 4\}$. After frame 0 is received without errors, it is looking for frames $\{1, 2, 3, 4, 5\}$ and so on. Hence after frames $0, 1, 2, 3, 4$ are received without errors, the receive window is $\{5, 6, 7, 0, 1\}$ because after 7 the sequence numbers wrap around. Now suppose that the ACKs for all the frames $0, 1, 2, 3, 4$ are received with errors at the source. Then they are all ignored and a timeout occurs at the source resulting in a retransmission of frames 0 through 4. If frame 0 is received correctly at the destination, then it will accepted. However, the destination thinks this is the ninth frame which also has sequence number 0 and fails to recognize that it is copy of the first frame. So duplicate frames are passed onto the network layer. This can be avoided if the window size is restricted to a maximum value of $N = 2^{3-1} = 4$. Then in the example just discussed frames $0, 1, 2, 3$ are sent initially and if they are received correctly the receive window is $\{4, 5, 6, 7\}$. So retransmissions of frames $0, 1, 2, 3$ due to lost ACKs will be rejected by the destination.

4.2.3 Performance Analysis of ARQ Protocols

Now that we have discussed three different ARQ protocols, we need a way to quantify their performance. This will not only tell us which ARQ protocol is better but also help us understand the cost benefit of using a lower complexity ARQ protocol. In general, a *performance metric* is defined as a mapping from the set of solutions to a problem to a numerical quantity. In the present context, the problem is achieving reliable communication over a noisy channel and the set of solutions contains different ARQ protocols. The performance metric we will choose is called *throughput efficiency* which is defined as the following.

$$\text{Throughput Efficiency} = \frac{\text{Throughput}}{\text{Data Rate}}$$

As discussed before, throughput of a communication protocol is defined as the following.

$$\text{Throughput} = \frac{\text{Number of information bits communicated}}{\text{Time taken to communicate the information bits}}$$

In the context of ARQ protocols, the time taken to communicate a frame depends on the number of times a frame or an ACK is corrupted. For example, in SW ARQ the time taken to transfer a frame when both the frame and the corresponding ACK are received without errors is $T_1 = T_f + T_a + T_p + 2\tau$. If the frame is corrupted once or the ACK is corrupted once, the time taken is $T_t + T_1$ where T_t is the timeout duration. So we get a different value of throughput for different situations. The solution is to model the time taken to communicate the frame as a random variable and calculate the throughput using the expected value of time taken. So for ARQ protocols we are defining throughput as

$$\text{Throughput} = \frac{\text{Number of information bits communicated}}{\text{Expected value of the time taken to communicate the information bits}}$$

Analysis of SW ARQ

We will analyse SW ARQ under the assumption that the frame size is fixed and the destination always has a frame to send. The parameters required in the analysis are given in Table 4.2. Let X be the random variable which represents the time taken to transfer n information bits from source to the destination without errors using SW ARQ. If the frame and ACK are received without errors, $X = T_a + T_p + T_f + 2\tau$ as shown in Figure 4.8. Note that we have included the time it takes for the ACK reach the source in our estimate of the time needed to transfer

Number of information bits in frame	n
Number of CRC bits in frame	k
Frame size in bits	$n+k$
Number of information bits in ACK	m
Number of CRC bits in ACK	k
ACK size in bits	$m+k$
Data Rate	$\frac{1}{T}$
Frame transmission time	$T_f = (n+k)T$
ACK transmission time	$T_a = (m+k)T$
Propagation delay	τ
Processing delay	T_p
Timeout duration	T_t
Probability of errors in frame	P_{FE}
Probability of errors in ACK	P_{AE}

Table 4.2: Parameters used in the analysis of SW ARQ

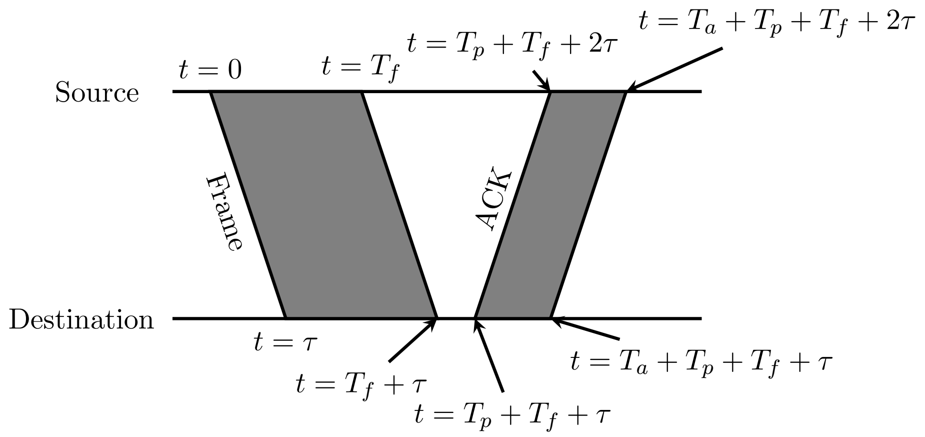


Figure 4.8: Illustration of frame and ACK transmission delays in SW ARQ

n bits from source to destination. This is because the source cannot send the next frame until the ACK is received from the destination. Let $T_1 = T_a + T_p + T_f + 2\tau$. Then $X = T_t + T_1$ if the first transmission of the frame incurs errors and the second frame transmission and ACK transmission are error-free or the first ACK transmission incurs errors and the second frame transmission and ACK transmission are error-free. These situations are illustrated in Figures 4.4b and 4.4c. Similarly, $X = lT_t + T_1$ if the first l transmissions of a frame are not acknowledged and the $(l+1)$th transmission is acknowledged $(l = 0, 1, 2, \ldots)$. Since X cannot take any other values we have

$$E[X] = \sum_{l=0}^{\infty} (lT_t + T_1) \Pr[X = lT_t + T_1]. \tag{4.10}$$

We need to calculate the probability of X being equal to $lT_t + T_1$ to be able to calculate $E[X]$. We will assume that the errors which occurs in two different transmissions are independent irrespective of whether the transmissions are frames or ACKs. First consider $l = 0$.

$$\begin{aligned}
\Pr[X = T_1] &= \Pr\left[(\text{Frame has no errors}) \cap (\text{ACK has no errors})\right] \\
&= \Pr\left[\text{Frame has no errors}\right] \cap \Pr\left[\text{ACK has no errors}\right] \\
&= (1 - P_{FE})(1 - P_{AE})
\end{aligned}$$

Now let $l = 1$.

$$\begin{aligned}
\Pr[X = T_t + T_1] &= \Pr[\{(\text{First transmission of frame has errors}) \\
&\quad \cap (\text{Second transmission of frame has no errors}) \\
&\quad \cap (\text{ACK has no errors})\} \\
&\quad \bigcup \{(\text{First transmission of frame has no errors}) \\
&\quad \cap (\text{ACK has errors}) \\
&\quad \cap (\text{Second transmission of frame has no errors}) \\
&\quad \cap (\text{ACK has no errors})\}] \\
&= \Pr[\{(\text{First transmission of frame has errors}) \\
&\quad \cap (\text{Second transmission of frame has no errors}) \\
&\quad \cap (\text{ACK has no errors})\}] \\
&\quad + \Pr[\{(\text{First transmission of frame has no errors}) \\
&\quad \cap (\text{ACK has errors}) \\
&\quad \cap (\text{Second transmission of frame has no errors}) \\
&\quad \cap (\text{ACK has no errors})\}] \\
&= \Pr[\text{First transmission of frame has errors}] \\
&\quad \times \Pr[\text{Second transmission of frame has no errors}] \\
&\quad \times \Pr[\text{ACK has no errors}] \\
&\quad + \Pr[\text{First transmission of frame has no errors}] \\
&\quad \times \Pr[\text{ACK has errors}] \\
&\quad \times \Pr[\text{Second transmission of frame has no errors}] \\
&\quad \times \Pr[\text{ACK has no errors}] \\
&= P_{FE}(1 - P_{FE})(1 - P_{AE}) + (1 - P_{FE})P_{AE}(1 - P_{FE})(1 - P_{AE}) \\
&= [P_{FE} + (1 - P_{FE})P_{AE}](1 - P_{FE})(1 - P_{AE})
\end{aligned}$$

where the second equality followed from the mutual exclusivity of the events on either side of the union and the third equality followed from our assumption that the errors in two different transmissions are independent.

By a similar argument we can show that

$$\Pr[X = lT_t + T_1] = [P_{FE} + (1 - P_{FE})P_{AE}]^l (1 - P_{FE})(1 - P_{AE})$$

for all $l = 0, 1, 2, 3, \ldots$. Substituting this into Equation (4.10), we get

$$\begin{aligned}
E[X] &= \sum_{l=0}^{\infty} (lT_t + T_1)[P_{FE} + (1 - P_{FE})P_{AE}]^l (1 - P_{FE})(1 - P_{AE}) \\
&= (1 - P_{FE})(1 - P_{AE}) \sum_{l=0}^{\infty} (lT_t + T_1)[P_{FE} + (1 - P_{FE})P_{AE}]^l
\end{aligned}$$

Since for $|x| < 1$, $\sum_{l=0}^{\infty} x^l = \frac{1}{1-x}$ and $\sum_{l=0}^{\infty} lx^l = \frac{x}{(1-x)^2}$, we get

$$\begin{aligned}
E[X] &= (1 - P_{FE})(1 - P_{AE}) \left[\frac{T_t[P_{FE} + (1 - P_{FE})P_{AE}]}{[1 - P_{FE} - (1 - P_{FE})P_{AE}]^2} + \frac{T_1}{[1 - P_{FE} - (1 - P_{FE})P_{AE}]} \right] \\
&= (1 - P_{FE})(1 - P_{AE}) \left[\frac{T_t[P_{FE} + (1 - P_{FE})P_{AE}]}{[(1 - P_{FE})(1 - P_{AE})]^2} + \frac{T_1}{(1 - P_{FE})(1 - P_{AE})} \right]
\end{aligned}$$

$$= \quad T_1 + \frac{T_t[P_{FE} + (1 - P_{FE})P_{AE}]}{(1 - P_{FE})(1 - P_{AE})}$$

A simple way to verify that we have not committed any major errors in our calculation is to look at the boundary cases. Suppose the channel is error-free. Then $P_{FE} = P_{AE} = 0$ and by substituting this into the above equation we get $E[X] = T_1$. This is consistent because in an error-free channel every frame is acknowledged in time T_1. This mean that X takes value T_1 with probability 1 and hence $E[X] = T_1$. Suppose that the channel is extremely noisy and $P_{FE} \to 1, P_{AE} \to 1$. Then $E[X] \to \infty$ which is expected since there will many retransmissions in a noisy channel and the time taken will be very large.

We will make the assumption that $P_{AE} \approx 0$ in order to simplify the expression for the throughput of SW ARQ. The other reason for this assumption is that analysis of GBN ARQ becomes very complicated without it and since we plan to compare the performance of the different ARQ protocols we want to make the same assumptions for all of them. This assumption of the probability of ACK error being small can be justified when the size of the ACK is small and the bit error rate (BER) of the channel is small. When a frame of length M bits is transmitted across a channel with bit error rate p, the probability no errors will be introduced in the frame is $(1-p)^M$ assuming that the events of error occuring in different bits are independent. Then the probability of errors being introduced in the frame is $1-(1-p)^M$. Now if p is small and M is small, $(1-p)^M$ will be close to 1 and $1-(1-p)^M$ will be close to 0. So if the ACK frame length is small and channel BER is small, P_{AE} can be approximated by 0. Then the expression for $E[X]$ becomes

$$E[X] \quad = \quad T_1 + \frac{T_t P_{FE}}{1 - P_{FE}}$$

Since the frame has n information bits, the throughput of SW ARQ is

$$\text{Throughput of SW ARQ} \quad = \quad \frac{n}{E[X]} = \frac{n}{T_1 + \frac{T_t P_{FE}}{1 - P_{FE}}}$$

Since the data rate is $\frac{1}{T}$, the throughput efficiency of SW ARQ is

$$\text{Throughput efficiency of SW ARQ} \quad = \quad \text{TE}_{SW} = \frac{\text{SW ARQ Throughput}}{\frac{1}{T}} = \frac{nT}{T_1 + \frac{T_t P_{FE}}{1 - P_{FE}}}$$

If we assume that $T_t = T_1$, the we

$$\begin{aligned} \text{TE}_{SW} \quad &= \quad \frac{nT}{T_1 + \frac{T_1 P_{FE}}{1 - P_{FE}}} = \frac{nT(1 - P_{FE})}{T_1} = \frac{nT(1 - P_{FE})}{T_f + 2\tau + T_p + T_a} \\ &= \quad \frac{n(1 - P_{FE})}{n + k + \frac{2\tau + T_p}{T} + m + k} \end{aligned}$$

where the last equality follows from the fact that $T_f = (n+k)T$ and $T_a = (m+k)T$. Thus the throughput efficiency decreases if the probability of frame error P_{FE} increases or if the number of check bits k increases or if the propagation delay τ increases or the processing delay T_p increases.

Analysis of GBN ARQ

Analysis of SR ARQ

4.3 Medium Access Control

As introduced in Section 1.2.3, multiple access channels are channels where the received signal at a destination node depends on the signal transmitted by several source nodes. The main challenge in multiple access channels is the problem of *channel allocation*, i.e. coordinating the usage of the multiple access channel among multiple source-destination pairs. The distributed algorithms which implement the channel allocation are called *medium access control (MAC) protocols* or *multiple access protocols.*

A *collision* is said to occur in a multiple access channel when more than the allowed number of sources send information at the same time resulting in a corruption of the signals received at one or more destinations. On the basis of whether collisions occur or not, MAC protocols can be classified into *random access* or *conflict-free* MAC protocols. In this section, we discuss some representative MAC protocols from each class.

4.3.1 Conflict-free MAC protocols

In conflict-free MAC protocols, collisions are completely avoided by allocating the channel access to sources in a predetermined manner. The two most important conflict-free protocols are *frequency division multiple access (FDMA)* and *time division multiple access (TDMA).*

FDMA

In FDMA, the entire channel banwidth is divided into bands each which is reserved for use by a single source. So if there are N sources and the channel bandwidth is W, each source is allocated a $\frac{W}{N}$ chunk of bandwidth for its own exclusive use. Since each source transmits only in its frequency band, collisions are completely avoided. FDMA is a simple and efficient solution to the channel allocation problem when the number of sources is a small constant and each of them always has some information to transmit. FDMA also does not require any coordination among the sources once the channel band allocation has been fixed greatly simplifying the system implementation. But FDMA is inefficient when the number of sources is large and varying or when the sources send information in a bursty manner. If the bandwidth is divided into N bands with each band allocated to a source, more than N sources cannot use the channel. This problem is aggravated if the users who have been allocated frequency bands send information intermittently. Then the channel is underutilized even though there are sources who want to transmit information through it. Even if the number of sources is small and constant, FDMA is inefficient when the source traffic is bursty because bandwidth allocated to sources which do not have any information to send at a particular time could have been used to increase the transmission rate of the sources which do have information to send.

Suppose the channel of bandwidth W can sustain a data rate of R bits/s. Then the per source data rate when FDMA is used is $\frac{R}{N}$. Suppose all the frames sent by the sources are of length equal to L bits. Then the service time of each frame is $\frac{NL}{R}$ seconds. We will assume that each source generates frames according to a Poisson process with rate λ frames per second. We will also assume that the channel is error-free to isolate the effect of the queueing delay of the frames from the delays caused by ARQ retransmissions. Then the FDMA system can be viewed as an $M/G/1$ queueing system (discussed in Section 5.4) with mean service time $\overline{X} = \frac{NL}{R}$ and the second moment of the service time $\overline{X^2} = \frac{(NL)^2}{R^2}$. In fact, it is an $M/D/1$ queueing system. The average time spent by a frame (which is the customer here) in the system is given by the following equation where $\rho = \frac{\lambda}{\mu} = \lambda\overline{X}$.

$$T = \overline{X} + \frac{\lambda\overline{X^2}}{2(1-\rho)} = \overline{X} + \frac{\rho\overline{X}}{2(1-\rho)} = \overline{X}\left[1 + \frac{\rho}{2(1-\rho)}\right] = \frac{NL}{R}\left[\frac{2-\rho}{2(1-\rho)}\right]$$

If the frame was sent using the whole channel bandwidth, the transmission time is given by $\frac{L}{R}$. Normalizing the above equation by this value we get the normalized delay experienced by a frame entering the system as

$$\hat{T}_{FDMA} = \frac{N(2-\rho)}{2(1-\rho)}$$

which gives the delay-throughput characteristic of FDMA where ρ can be interpreted as the throughput. We see that the delay increases linearly with the number of sources N. It also grows without bound when ρ approaches one.

TDMA

In TDMA, the time axis is divided into slots and each slot is allocated to a source. Each source transmits only during the time slot reserved for its use avoiding the possibility of collisions. If there are N sources, the collection of N slots in which each source transmits exactly once is called a *cycle*. Contrary to FDMA, TDMA requires some amount of synchronization among the sources in terms of the location and duration of the time slots when transmissions can occur. Pure TDMA suffers from the same problem of underutilization as FDMA when the source traffic is bursty.

Once again we assume that all the frames generated by the sources have equal length (L bits). We assume that the channel can support a data rate equal to R bits/s. Then the transmission time of a frame is $\frac{L}{R}$. We assume that the slot duration is equal to $\frac{L}{R}$, the time required to transmit one frame. Once again each source is assumed to generate frames according to a Poisson process with rate λ frames per second. The time spent by a frame in the system, i.e. the delay experienced by a frame has three components.

- Time between generation of the frame and end of the current cycle
- Time spent waiting for other frames which were generated before it to be transmitted
- The frame transmission time

To calculate the average delay experienced by a frame we have to invoke the theory of the $M/G/1$ queueing system with vacations. In such a system, the server takes a "vacation" whenever it finds that there are no customers to be served. If customers arrive during a vacation, they are not served until the vacation is complete. If no customers arrive during a vacation, the server takes another vacation. The formula for the average time spent by a customer in such a system is given by

$$T = \frac{L}{R} + \frac{\lambda\overline{X^2}}{2(1-\rho)} + \frac{\overline{V^2}}{2\overline{V}}$$

where $\overline{V}$ and $\overline{V^2}$ are the average vacation time and the second moment of the vacation time respectively. The TDMA system can be interpreted as such a system because of the cycle structure. Suppose there are N sources and suppose source 1 has no frames to send when its slot arrives, then the next opportunity for source 1 appears only after $\frac{NL}{R}$ seconds, i.e. after N slots. This can be viewed as a vacation of duration $\frac{NL}{R}$ because a frame generated by source 1 during this time has to wait till the end of the cycle to be transmitted. For TDMA, $\overline{V} = \frac{NL}{R}$, $\overline{V^2} = \frac{(NL)^2}{R^2}$, $\overline{X} = \frac{NL}{R}$ and $\overline{X^2} = \overline{X}^2$. Then the normalized delay for the TDMA system is given by

$$\hat{T}_{TDMA} = \frac{T}{\frac{L}{R}} = 1 + \frac{N\rho}{2(1-\rho)} + \frac{N}{2} = 1 + \frac{N}{2(1-\rho)}$$

We see that the difference between the normalized delays in TDMA and FDMA is given by

$$\hat{T}_{FDMA} - \hat{T}_{TDMA} = \frac{N(2-\rho)}{2(1-\rho)} - 1 - \frac{N}{2(1-\rho)} = \frac{N}{2} - 1.$$

Thus the average delay experienced by a frame in FDMA is greater than the average delay experienced by a frame in TDMA when $N \geq 2$. Also, the difference grows linearly with N and is independent of the arrival rate λ.

4.3.2 Random Access MAC Protocols

In the conflict-free MAC protocols discussed in the previous section, collisions are completely avoided by design. But there is wastage of channel resources when the traffic is bursty. Random access MAC protocols prevent this wastage in the case of bursty traffic when the traffic load is low at the cost of occassional collisions between source transmissions. Colliding frames would have to be retransmitted until they are received correctly. When the traffic load is high, frequent collisions occur when random access MAC protocols are used reducing their efficiency.

For the random access MAC protocols discussed below, we assume that all the sources accessing the channel are one hop away from each other. Thus the channel is a *broadcast* channel, i.e. all the sources can hear the transmissions of all the other sources. We also assume that if a collision occurs, i.e. the transmissions of two or more sources overlap in time, none of the intended destinations receive the transmitted frames correctly. We assume that the only errors experienced by the frames are due to collisions, i.e. the channel noise is negligible.

ALOHA

In the ALOHA protocol, sources transmit whenever they have a frame to send. If a source's frame transmission suffers a collision, the source detects the collision either by listening while transmitting or by the absence of an acknowledgement. Once collision has been detected, the source waits for a random amount of time before retransmitting the frame. This random waiting time is used to avoid the situation where the same frames keep colliding indefinitely.

We will now perform an approximate analysis to estimate the throughput of the ALOHA protocol where throughput is defined as the fraction of the frame transmissions which are successful. Assume that all the frames generated by the sources are of equal length and require T_f seconds to be transmitted. Assume that there is an infinite population of sources which cumulatively generate frames according to a Poisson distribution with mean rate of λ frames per T_f seconds, i.e. λ frames per frame transmission time. The infinite population assumption is made to ensure that the frame generation rate remains constant even if some sources are involved in collisions and are thus not generating new frames. If $\lambda > 1$, the sources are generating frames faster than the channel can accommodate even if there were no collisions. In such a system, nearly every frame will suffer a collision. So a feasible system has to have $0 < \lambda < 1$.

Sources not only transmit newly generated frames but also transmit frames which were previously involved in a collision. We assume that the *offered load* on the channel is also Poisson distributed with mean rate G frames per frame transmission time where $G > \lambda$. Then the throughput S is the offered load G times the probability P_0 of a successful transmission, i.e. $S = GP_0$.

A frame transmitted at time t_0 will not suffer a collision if no frames are transmitted in the interval $[t_0 - T_f, t_0 + T_f]$. Since the offered load is Poisson distributed with mean rate G frames per T_f seconds, the probability that k frames are transmitted in the interval of size $2T_f$ is given by

$$\Pr\{k \text{ frames transmitted in } 2T_f \text{ seconds}\} = \frac{(2G)^k e^{-2G}}{k!}$$

Thus $P_0 = \Pr\{0 \text{ frames transmitted in } 2T_f \text{ seconds}\} = e^{-2G}$ and the throughput of ALOHA protocol is given by

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

The maximum throughput occurs when $G = 0.5$ and is $S = \frac{1}{2e} \approx 0.184$. So frames are transmitted successfully only 18% of the time.

Slotted ALOHA

Slotted ALOHA is a simple way of achieving double the througput of the ALOHA protocol. It is similar to ALOHA except that the time axis is divided into slots and sources are allowed to transmit frames only at the beginning of slots. This reduces the vulnerability duration for a frame transmission by half compared to pure ALOHA.

We give an approximate analysis which gives us the correct througput of slotted ALOHA without resorting to the complex theory of regenerative processes. A frame transmitted at a slot boundary beginning at time t_0 will not suffer a collision if no other frames are transmitted in that slot, i.e. no frames were scheduled for transmission in the interval $[t_0 - T_f, t_0]$ where we have assumed that a slot duration is equal to a frame transmission time. If the offered load is again assumed to be Poisson distributed with mean rate G frames per T_f seconds, the probability that k frames are scheduled for transmission in an interval of size T_f is given by

$$\Pr\{k \text{ frame transmissions scheduled in } T_f \text{ seconds}\} = \frac{(G)^k e^{-G}}{k!}$$

Thus the probability of successful frame transmission P_0 is equal to the probability that 0 frame transmissions were scheduled in T_f seconds which is e^{-G} and the throughput of ALOHA protocol is given by

$$S = GP_0 = Ge^{-G}$$

The maximum throughput occurs when $G = 1$ and is $S = \frac{1}{e} \approx 0.368$. So frames are transmitted successfully 37% of the time which is double that of ALOHA.

Carrier Sense Multiple Access (CSMA) Protocols

In the ALOHA protocols, the sources transmit without regard to whether the channel is being used at that instant and a maximum throughput of $\frac{1}{e}$ is achieved. A higher throughput can be achieved by *carrier sense multiple access (CSMA)* protocols if the sources have ability to listen to the channel and detect ongoing transmissions, i.e. they have the ability to sense the carrier.

The simplest CSMA protocol is called *1-persistent CSMA*. In this protocol, when a source has a frame to send it listens to channel to see if any other source is transmitting. If the channel is idle, it will transmit its frame. If the channel is busy, the source keeps listening to the channel until it becomes idle and then transmits its frame. If a collision occurs, the source waits a random amount of time and then starts listening to the channel again. The protocol is called 1-persistent because it allows sources to transmit with probability 1 whenever they find the channel to be idle.

In 1-persistent CSMA, collisions will definitely occur at the end of a frame transmission if more than one source has been waiting for the channel to become idle. A solution to this problem is the *nonpersistent CSMA*. In this protocol, a source transmits its frame if it finds the channel to be idle. If the channel is busy, the source does not continuously listen to the channel. It waits for a random amount of time and starts listening to the channel. This protocol reduces the number of collisions as compared to 1-persistent CSMA but results in longer delays being experienced by a frame.

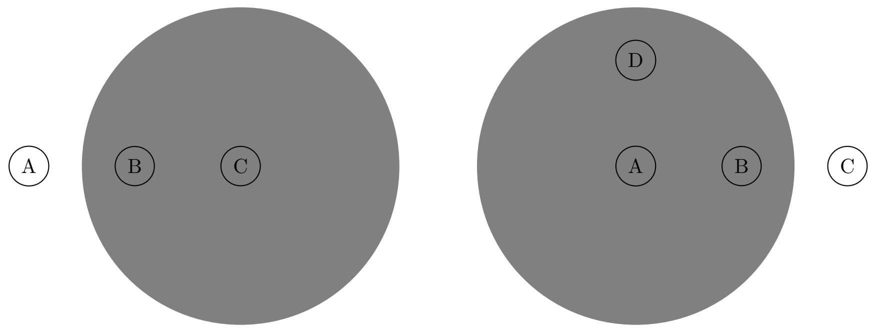
The *p-persistent CSMA* protocol is used in slotted channels. In this protocol, when the source has a frame to send it listens to the channel. If the channel is idle, the source transmits its frame in the current slot with a probability p and defers until the next slot with probability $q = 1 - p$. If the channel is idle in the next slot as well, the source either transmits or defers again with probabilities p and q respectively. This process continues until either the frame has been successfully transmitted or another source has started transmitting. In the latter situation, the source reacts as if a collision has occurred, i.e. it waits for a random amount of time and starts listening to the channel.

The propagation delay plays an important role in the functioning of CSMA protocols. Suppose a source listens to the channel and decides to transmit on finding it idle. It can happen that a second source starts listening to the channel just after the first source starts transmitting and declares that the channel is idle because the signal from the first source has not reached it yet. It can then proceed to transmit its own frame resulting in a collision. So it is necessary that the propagation delay be small, otherwise the above situation will occur more frequently.

CSMA/CD Protocols

In *carrier sense multiple access with collision detection (CSMA/CD)* protocols, sources abort their transmissions as soon as they detect a collision. So if two sources start transmitting at the same time after finding the channel to be idle, their frames will collide but the sources will abort the frame transmissions as soon as they detect a collision instead of finishing the frame transmissions. Sources involved in a collision wait for a random amount of time before listening to the channel.

To make collision detection possible the source has to listen to the channel while transmitting. This makes CSMA/CD a half-duplex system because the receiver subsystem is in constant use listening for collisions even during transmission. Furthermore, a source has to listen for collisions



(a) The hidden node problem (b) The exposed node problem

Figure 4.9: Illustration of the hidden node and exposed node problems.

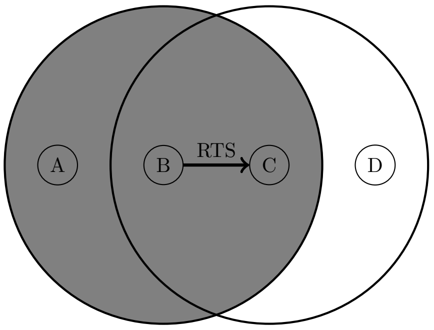
for a duration equal to the maximum round-trip time of the multiple access channel before it is sure that it has captured the channel. Suppose the maximum propagation delay between sources is τ, i.e. the maximum round-trip time is 2τ. Suppose a source starts transmitting a frame at time t_0. Then a second source which is at a propagation delay τ from the first one will hear the transmission only at time $t_0 + \tau$. If the second source starts transmitting a frame at time $t_0 + \tau - \epsilon$, it will detect the collision at time $t_0 + \tau$ and abort its transmission. But the first source will hear the short burst sent by the second source only at $t_0 + 2\tau - \epsilon$.

CSMA/CA Protocols

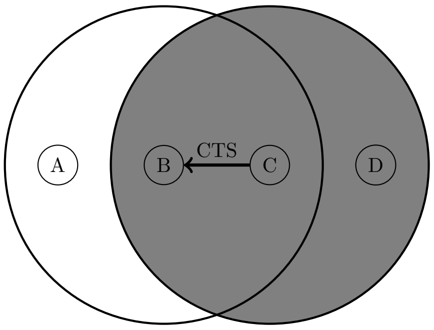
In wireless multiple access channels, it is sometimes not possible for a source to detect ongoing transmissions at faraway sources due to the path loss. In these channels, all assumption of all sources being one hop away from each other no longer holds. For such channels, *carrier sense multiple access with collision avoidance (CSMA/CA)* protocols are used.

In addition to coordinating the channel access, these protocols solve what is called the *hidden node problem*. This is illustrated in Figure 4.9a where the grey circle represents the transmission range of the node C. The node A wants to send a frame to B. Node C is transmitting a frame to node B but its transmission is not heard by node A because it out of range of A. So when node A listens to the channel it decides that the channel is idle and proceeds to transmit a frame to B resulting in a collision at node B. There is a second problem called the *exposed node problem* which cannot be completely solved by these protocols. This is illustrated in Figure 4.9b. In this case, node A is transmitting to node D and node B lies within its transmission range. Node B wants to send a frame to node C which it can do even in the presence of node A's transmission because node C lies outside the transmission range of node A. But when node B listens to the channel it hears A's transmission, decides that the channel is busy and refrains from sending a frame to C.

The main idea behind CSMA/CA is that the source requests the destination to send a short frame informing all the nodes in the destination's neighborhood of an impending transmission which should not be interfered with. Suppose a node B wants to send a frame to node C it sends an *Request to Send (RTS)* frame to C. This short frame contains the length of the longer data frame which will follow. This is illustrated in Figure 4.10a where the grey circle represents the transmission range of node B. Node C responds by sending a *Clear to Send (CTS)* frame which also contains the length of the upcoming data frame. Once the CTS is received by node B, it begins to transmit the data frame. The main consequence of the RTS-CTS mechanism is that



(a) Node B sending RTS to node C



(b) Node C sending CTS to node B

Figure 4.10: Illustration of the RTS-CTS mechanism

the nodes which over the RTS and CTS refrain from transmitting during the frame transmission from node B to node C. For instance, when node D hears the CTS frame it reads the length of the upcoming data frame and defers transmission until the data frame is transmitted completely. Since D does not hear the RTS sent by B it can infer from the CTS that a hidden node (node B in this case) is trying to send data to node C. Similarly, when node A hears the RTS sent by B but not the CTS sent by C it infers that it is itself an exposed node and is free to transmit while the transmission from B to C is ongoing. In spite of the RTS-CTS mechanism, collisions can occur, for example, when two nodes transmit an RTS at the same time. When collisions occur, the nodes involved in the collision wait a random amount of time and retransmit.

Chapter 5

Queueing Theory

Any system in which consumers have to wait to gain access to a finite-capacity resource can be called a *queueing system*. The finite-capacity resource is called a *server* and the consumers are typically referred to as *customers*. A *queue* of customers results when their service requirements cannot be immediately met by the server. *Queueing theory* is the study of the phenomena of waiting and serving at a queue. The dynamics of a queueing system primarily depends on two aspects: the arrival behavior of the customers and the serving capability of the server. Often the service rate of the server is constant and it is the customers who each require a different amount of service from it. In queueing theory, the customer arrival process is characterized by the probability distribution of the *interarrival times* of customers which is given by

$$A(t) = \Pr\left[\text{Time between customer arrivals} \leq t\right].$$

The usual assumption is that the interarrival times of customers are independent, identically distributed (iid) random variables. Hence $A(t)$ completely characterizes the arrival process. Similarly, the serving capability of the server is characterized by the probability distribution of the *service time* which is given by

$$B(t) = \Pr\left[\text{Service time} \leq t\right].$$

Additional quantities which describe the capabilities of the server are the maximum number of customers who can be accomodated by the server in a queue (this is usually assumed to be infinite) and the number of customers who can be serviced simultaneously (this is assumed to be one for now).

The performance metrics used to characterize the quality of a queueing system are the waiting time for a customer, the number of customers waiting in the system, the length of a busy period (the time during which the server is busy) and the length of an idle period (the time during which the server is idle). These metrics are random variables and vary from one realization of the system to another. So their mean and variance are used to quantify the performance of the queueing system.

In the next section, we introduce some basic notation which will serve to illuminate the structure of a queueing system.

5.1 Preliminaries

5.1.1 Kendall's notation

Different types of queueing systems are described by convenient shorthand called *Kendall's notation.* It consists of a five-part descriptor $A/B/m/K/L$ where

- A describes the type of customer arrival process. When the arrival process is *Poisson process*, A is replaced by M which stands for memoryless. We will see that the interarrival times for a Poisson process are exponentially distributed. A process with deterministic interarrival time is denoted by$D$ and one with any general distribution of interarrival times is denoted by G.
- B describes the type of service time distribution. It can takes values M, G and D corresponding to exponentially distributed, generally distributed and deterministic service times.
- m indicates the number of servers, i.e. the number of customers who can be serviced simultaneously.
- K denotes the queueing system's storage capacity, i.e. the maximum number of customers who can be present in the system.
- L denotes the size of the customer population.

If K and L are assumed to be infinite, they are omitted from the description and we are left with a three-part descriptor $A/B/m$. For example, the $M/M/1$ queueing system consists of a single server with infinite storage capacity servicing an infinite population of customers who arrive with interarrival times which are exponentially distributed and experience service times which are also exponentially distributed. The assumptions of infinite storage capacity and population help make the analysis simple. The analytical results obtained through these assumptions apply to practical systems when the storage capacity and customer population are large.

5.1.2 Queue state variables

We will denote the nth customer who arrives at a queue by C_n and the number of customers in the system at time t by $N(t)$. Let $U(t)$ denote the amount of unfinished work in the system at time t, i.e. the time required to service all the customers present in the system at time t. When $U(t) > 0$, the system is busy and the system is idle when $U(t) = 0$. The busy and idle times of the system are of interest to the system designer. These can be characterized in terms of the interarrival times and service times of the customers.

Let t_n denote the arrival time of the nth customer C_n. Then the interarrival time between C_{n-1} and C_n is defined as $\tau_n = t_n - t_{n-1}$. If we assume that the interarrival times are iid and their distribution is $A(t)$, we have $\Pr[\tau_n \le t] = A(t)$.

Let x_n denote the service time for C_n. If we assume the service times are iid and distributed according to $B(x)$, then $\Pr[x_n \le x] = B(x)$. But the nth customer receives service from the server after it has waited in the queue for some time which we define as the waiting time w_n. Thus the total time spent by C_n is the sum of this waiting time and the service time which we define as his system time $s_n = w_n + x_n$.

5.1.3 Poisson process

Consider a stochastic process $\{A(t)|t \geq 0\}$ which takes nonnegative integer values. Suppose the value of $A(t)$ is equal to the number of customer arrivals in the interval $[0, t]$ and that the number of arrivals in disjoint intervals are independent. Then $\{A(t)|t \geq 0\}$ is called a *Poisson process* with rate λ if the number of arrivals in any interval of length τ is Poisson distributed with parameter $\lambda\tau$, i.e. for all $t, \tau > 0$

$$\Pr\{A(t+\tau) - A(t) = n\} = e^{-\lambda\tau}\frac{(\lambda\tau)^n}{n!}, \quad n = 0, 1, 2, \ldots$$

The expectation or average value of the number of arrivals in an interval of length τ turns out to be $\lambda\tau$ which leads us to the interpretation of the λ as the arrival rate.

An important property of the Poisson process is that interarrival times are independent and exponentially distributed. So if t_n is the arrival time of the nth customer C_n, then the interarrival time between C_{n-1} and C_n, $\tau_n = t_n - t_{n-1}$ has probability distribution

$$\Pr[\tau_n \leq s] = 1 - e^{-\lambda s}, \text{ for } s \geq 0.$$

The corresponding probability density function is given by $p(\tau_n) = \lambda e^{-\lambda\tau_n}$. Also, the sequence of random variables $\{\tau_n\}$ are mutually independent. The mean and variance of the interarrival time turn out to be $\frac{1}{\lambda}$ and $\frac{1}{\lambda^2}$, respectively.

A unique characteristic of the exponential distribution is its *memoryless* character which can be expressed as

$$\Pr\{\tau_n > r + t | \tau_n > t\} = \Pr\{\tau_n > r\}, \text{ for } r \geq 0.$$

This means that the additional time before the next arrival is independent of the time which has already passed waiting for it. This can be proved in the following manner.

$$\Pr\{\tau_n > r + t | \tau_n > t\} = \frac{\Pr\{\tau_n > r + t\}}{\Pr\{\tau_n > t\}} = \frac{e^{-\lambda(r+t)}}{e^{-\lambda t}} = e^{-\lambda r} = \Pr\{\tau_n > r\}$$

In fact, the exponential distribution is the only probability distribution which has the memoryless property.

Using the fact that $e^{-\lambda\delta} = 1 - \lambda\delta + (\lambda\delta)^2/2 - \cdots$, the probability of the number of arrivals due to a Poisson process in an interval of size δ can be expressed as

$$\begin{aligned}
\Pr\{A(t+\delta) - A(t) = 0\} &= 1 - \lambda\delta + o(\delta) \\
\Pr\{A(t+\delta) - A(t) = 1\} &= \lambda\delta + o(\delta) \\
\Pr\{A(t+\delta) - A(t) \geq 2\} &= o(\delta)
\end{aligned}$$

where $o(\delta)$ is any function of δ which satisfies

$$\lim_{\delta \to 0} \frac{o(\delta)}{\delta} = 0$$

If k independent Poisson processes $A_1, A_2, \ldots, A_k$ are merged into a single process $A = \sum_{i=1}^{k} A_i$, the process A is also Poisson with rate equal to the sum of the k rate of the constituent processes.

5.1.4 Markov chains

Consider a stochastic process $\{X_n : n = 0, 1, 2, \ldots\}$ that takes on a finite or countable number of possible values. Without loss of generality, this set of possible values for the process will be denoted by the set of nonnegative integers $\{0, 1, 2, \ldots\}$. If $X_n = i$ for $i \geq 0$, the process is said to be in state i at time n. A stochastic process $\{X_n : n = 0, 1, 2, \ldots\}$ is called a *Markov chain* if

$$\Pr\{X_{n+1} = j | X_n = i, X_{n-1} = i_{n-1}, \ldots, X_1 = i_1, X_0 = i_0\} = \Pr\{X_{n+1} = j | X_n = i\} = P_{ij},$$

for all states $i_0, i_1, \ldots, i_{n-1}, i, j$ and all $n \geq 0$. The interpretation is that the conditional distribution of any future state X_{n+1} given the past states $X_0, X_1, X_2, \ldots, X_{n-1}$ and the present state X_n, is independent of the past states and depends only on the present state.

If the process is in state i, P_{ij} is the probability that it will be in state j next and is called the transition probability from state i to state j. Note that P_{ij} does not depend on n, i.e. the transition probabilities are the same no matter when the process is in a particular state. Since the transition probabilities are nonnegative and the process has to make a transition to some state, we have $P_{ij} \geq 0$ for all $i, j \geq 0$ and

$$\sum_{j=0}^{\infty} P_{ij} = 1, \qquad i = 0, 1, 2, \ldots$$

5.2 Little's Theorem

When the average arrival rate exceeds the average service rate, the queue length becomes unbounded resulting in eventual system failure. The interesting case is when the average arrival rate is less than the average service rate. In this case, in order to predict the average delay or the average number of customers in the queueing system we require detailed statistical information regarding the customer arrival times and their service times. However, for queueing systems which reach a steady state there exists a simple relationship between the average delay and the average number of customers called *Little's theorem.*

Let $\alpha(t)$ denote the number of customers who have entered the queueing system in the interval $(0, t)$. Let $\beta(t)$ denote the number of customers who depart the system after completion of their service in the interval $(0, t)$. An observer at the input of the queueing system would observe $\alpha(t)$ customers entering the system in the given interval and an observer at the output of the queueing system would observe $\beta(t)$ customers departing the system in the given interval. Then at any time t, the number of customers in the system $N(t)$ is given by

$$N(t) = \alpha(t) - \beta(t).$$

If the functions $\alpha(t)$ and $\beta(t)$ are plotted as functions of time, the area between them represents *the total time all customers have spent in the system* (measured in customer-seconds) during the interval $(0, t)$. Let $\gamma(t)$ denote this area. Then

$$\gamma(t) = \sum_{i=1}^{\alpha(t)} s_i$$

where s_i is the system time (sum of waiting time and service time) of the ith customer. Let λ_t be the average arrival rate in the interval $(0, t)$. Then

$$\lambda_t = \frac{\alpha(t)}{t}.$$

If we define T_t as the average system time per customer over the interval $(0, t)$, we get

$$T_t = \frac{\gamma(t)}{\alpha(t)}$$

If we define $\bar{N}_t$ as the average number of customers in the queueing system in the interval $(0, t)$, we can calculate it as the total number of customer-seconds accumulated divided by the interval length t

$$\bar{N}_t = \frac{\gamma(t)}{t}$$

From the above three equations we see that

$$\bar{N}_t = \lambda_t T_t$$

Now if we assume that the queueing system reaches a steady state, i.e. the following limits exist

$$\lambda = \lim_{t\to\infty} \lambda_t$$
$$T = \lim_{t\to\infty} T_t$$

then the limit $\lim_{t\to\infty} \bar{N}_t$ will also exist. This limit is the average number of customers in the system $\bar{N}$ which now satisfies the equation

$$\bar{N} = \lambda T.$$

This equation is called Little's theorem. It states that, for a queueing system which reaches a steady state, the average number of customers is equal to the product of the average arrival rate of the customers to the system and the average time spent in that system.

In the above proof, we have not made any specific assumptions regarding the interarrival time distribution or the service time distribution. Also, we have not precisely defined the boundary around the queueing system. So if we consider a system which consists of only the queue and not the server, Little's theorem would state that

$$\bar{N}_q = \lambda W.$$

where $\bar{N}_q$ is the average number of customers waiting in the queue and W is the average time spent by a customer waiting in the queue. If we had considered only the server and not the queue, we would get

$$\bar{N}_s = \lambda \bar{x}.$$

where $\bar{N}_s$ is the average number of customers in the server and $\bar{x}$ is the average time spent by the customer in the server. It is always true that

$$T = \bar{x} + W$$

5.3 The $M/M/1$ Queueing System

The $M/M/1$ queueing system consists of a single server serving customers who arrive according to a Poisson process with rate λ, i.e. their interarrival times are exponentially distributed. The service times of the customers are also exponentially distributed. The storage capacity of the system is assumed to be infinite and the population from which the customers arrive is also assumed to be infinite. From Little's theorem, we know that the average number of customers in the system is given by

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

where T is the average time spent by a customer in the system. If we can calculate the steady-state probabilities of the number of customers in the system,

$$p_n = \text{Probability of } n \text{ customers in the system, } n = 0, 1, 2, \ldots,$$

we can calculate the average number of customers $\bar{N}$ as

$$\bar{N} = \sum_{n=0}^{\infty} n p_n.$$

Then by applying Little's theorem, we can get the average time spent by a customer as

$$T = \frac{\bar{N}}{\lambda}.$$

Let $N(t)$ denote the number of customers present in the system at time. Due to the Poisson arrival process and memoryless nature of service times, $\{N(t)|t \geq 0\}$ forms a continuous-time Markov chain, i.e. the future numbers of customers depends on the past numbers only through the present number of customers in the system. However, the simpler theory of discrete-time Markov chains is sufficient to calculate the average number of customers in the system. The continuous-time Markov chain is discretized by considering the state of the system at times $\{k\delta : k \geq 0\}$ where δ is a small positive real number. Let $N_k = N(k\delta)$. Then $\{N_k|k = 0, 1, 2, \ldots\}$ is a discrete-time Markov chain with the same p_n's as the continous-time Markov chain. Note the p_n can be interpreted as the fraction of time in which the system has n customers. If δ is small enough, this fraction is preserved even after we discretize time.

5.3.1 State Transition Probabilities

The system is in state n at a particular time instant $k\delta$ if $N_k = n$ customers at time instant $k\delta$. So a transition from state i to state $i+1$ occurs if m customers arrive in the interval $I_k = (k\delta, (k+1)\delta]$ and there are $m - 1$ customer departures in the same interval where $m = 1, 2, 3 \ldots$. Let P_{ij} denote the transition probability from state i to state j. Then

$$P_{ij} = \Pr\{N_{k+1} = j | N_k = i\}$$

where the dependence of P_{ij} on δ is not shown for simplicity. Then we can show that

$$\begin{aligned}
P_{00} &= 1 - \lambda\delta + o(\delta) \\
P_{ii} &= 1 - \lambda\delta - \mu\delta + o(\delta), \qquad i \geq 1 \\
P_{i,i+1} &= \lambda\delta + o(\delta), \qquad i \geq 0 \\
P_{i,i-1} &= \mu\delta + o(\delta), \qquad i \geq 1 \\
P_{ij} &= o(\delta), \qquad i \text{ and } j \neq i, i+1, i-1.
\end{aligned}$$

where the $o(\delta)$ represents any function which satisfies $\lim_{\delta \to 0} \frac{o(\delta)}{\delta} = 0$.

Let us look at some of these transition probabilities in detail. P_{00} is the probability that the system transitions from state 0, i.e. $N_k = 0$ to $N_{k+1} = 0$. Since N_k is the number of customers in the system at time $k\delta$, such a state transition occurs if the number of customer arrivals is equal to the number of customer departures in the interval I_k, i.e.

$$P_{00} = \sum_{m=0}^{\infty} \Pr\{m \text{ customers arrive and all of them depart in the interval } I_k\ \} \tag{5.1}$$

First let us estimate the probability that 0 customers arrive in the interval I_k. Since the number of customers arriving in an interval has Poisson distribution, this is given by

$$\Pr\{\text{No customers arrive in } I_k\} = e^{-\lambda\delta}\frac{(\lambda\delta)^0}{0!} = e^{-\lambda\delta} = 1 - \lambda\delta + o(\delta)$$

Now let us estimate the probability of one customer arriving and departing in the interval I_k. Since the arrival and departure processes are memoryless, we can take $k = 0$ and just look at the interval $(0, \delta]$. This makes our calculations easier. The customer can arrive at any time $t \in (0, \delta]$. Given that the customer arrived at time t, the probability that she will depart before time δ is the probability that the service time is less than or equal to $\delta - t$. Since the service time is exponentially distributed with rate μ, the probability of it being less than or equal to $\delta - t$ is given by

$$\Pr\{\text{Service time} \leq \delta - t\} = 1 - e^{-\mu(\delta - t)}$$

The interarrival time has an exponential distribution which is a continuous distribution. So the event that the customer arrives at a time exactly equal to t is zero. The trick is to consider an infinitesimal interval from t to $t + dt$ and look at the event of a customer arriving in this interval. Such an event has probability $\lambda e^{-\lambda t}\,dt$ where $\lambda e^{-\lambda t}$ is the probability density function of the interarrival time. Since t can vary from 0 to δ, we can write the probability of one customer arriving and departing in the interval $(0, \delta]$ as

$$\begin{aligned}
\Pr\{\text{One customer arriving and departing in } (0,\delta]\} &= \int_0^{\delta} \lambda e^{-\lambda t}(1 - e^{-\mu(\delta - t)})\,dt \\
&= -e^{-\lambda t}\Big|_0^{\delta} - \lambda e^{-\mu\delta}\,\frac{e^{(\mu-\lambda)t}}{\mu - \lambda}\Big|_0^{\delta} \\
&= 1 - e^{-\lambda\delta} - \lambda e^{-\mu\delta}\frac{e^{(\mu-\lambda)\delta} - 1}{\mu - \lambda} \\
&= 1 - e^{-\lambda\delta} - \lambda\frac{e^{-\lambda\delta} - e^{-\mu\delta}}{\mu - \lambda} \\
&= 1 - e^{-\lambda\delta} - \lambda\frac{(\mu - \lambda)\delta + o(\delta)}{\mu - \lambda} \\
&= 1 - e^{-\lambda\delta} - \lambda\delta + o(\delta) \\
&= o(\delta)
\end{aligned}$$

Let A_m be the event of m customers arriving and D_m be the event of m customers departing in the interval $(0, \delta]$. For $m \geq 2$, the probability of m customers arriving and departing in the interval $(0, \delta]$ is given by

$$\begin{aligned}
\Pr\{A_m \cap D_m\} &= \Pr\{D_m | A_m\}\Pr\{A_m\} \\
&= \Pr\{D_m | A_m\}\, e^{-\lambda\delta}\frac{(\lambda\delta)^m}{m!} \\
&= o(\delta)
\end{aligned}$$

where the last equation follows from the fact that $\Pr\{D_m|A_m\} \leq 1$, $e^{-\lambda\delta} = 1 - \lambda\delta + o(\delta)$ and $m \geq 2$.

Substituting the calculated probabilities in Equation (5.1) we get

$$P_{00} = 1 - \lambda\delta + o(\delta)$$

When the system is in state $i \geq 1$, then the probability of zero customer arrivals and departures in the interval I_k is given as

$$\Pr\{\text{No customers arrive or depart in } I_k\} = e^{-\lambda\delta}e^{-\mu\delta} = 1 - \lambda\delta - \mu\delta + o(\delta) \qquad (5.2)$$

where the last equation is obtained by expanding the exponentials as power series in δ.

When the system is in state $N_k = 1$, the probability of no customer arrivals and one departure in the interval I_k is given simply as

$$\Pr\{\text{Zero customers arrive and one departs in } I_k|N_k = 1\} = e^{-\lambda\delta}(1 - e^{-\mu\delta}) = \mu\delta + o(\delta) \quad (5.3)$$

When the system is in state $N_k > 1$, the event of no customer arrivals and one departure in the interval I_k is the event that exactly one customer departs and no new customer arrives. So after one customer departs the next customer whose service starts should not depart, i.e. his service should not complete. As before, we will look at the interval $(0, \delta]$ instead of I_k. The probability of no arrivals in this interval is just $e^{-\lambda\delta}$. If a customer departure occurs at $t \in (0, \delta]$, the service time of the subsequent customer needs to be larger than $\delta - t$. Since the service time is exponentially distributed with rate μ, the probability of it being larger than $\delta - t$ is given by

$$\Pr\{\text{Service time} > \delta - t\} = e^{-\mu(\delta - t)}$$

Considering an infinitesimal interval from t to $t + dt$, the probability of a customer completing service in this interval is $\mu e^{-\mu t}\, dt$. Since t can vary from 0 to δ, we can write the probability of exactly one customer departure and no arrivals in the interval $(0, \delta]$ as

$$\begin{aligned}\Pr\{\text{Zero customers arrive and one departs in } I_k|N_k > 1\} &= \int_0^\delta e^{-\lambda\delta}\mu e^{-\mu t}e^{-\mu(\delta - t)}\, dt \\ &= e^{-\lambda\delta}\mu e^{-\mu\delta}\int_0^\delta dt \\ &= \mu\delta e^{-(\lambda+\mu)\delta} \\ &= \mu\delta + o(\delta) \qquad (5.4)\end{aligned}$$

So the probability of zero customers arriving and one departing given there is at least one customer in the system is

$$\begin{aligned}&\Pr\{\text{Zero customers arrive and one departs in } I_k|N_k \geq 1\} \\ &= \Pr\{\text{Zero customers arrive and one departs in } I_k|N_k = 1\}\Pr\{N_k = 1|N_k \geq 1\} \\ &\quad + \Pr\{\text{Zero customers arrive and one departs in } I_k|N_k > 1\}\Pr\{N_k > 1|N_k \geq 1\} \\ &= [\mu\delta + o(\delta)]\left(\Pr\{N_k = 1|N_k \geq 1\} + \Pr\{N_k > 1|N_k \geq 1\}\right) \\ &= \mu\delta + o(\delta) \qquad (5.5)\end{aligned}$$

Similarly, the probability of one customer arrival and no departures in the interval can be calculated as

$$\Pr\{\text{One customer arriving and none departing in } I_k\} = \lambda\delta + o(\delta) \qquad (5.6)$$

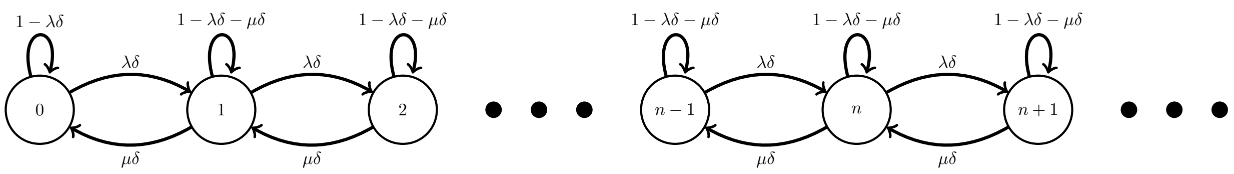

$1-\lambda\delta$
$1-\lambda\delta-\mu\delta$
$1-\lambda\delta-\mu\delta$
$1-\lambda\delta-\mu\delta$
$1-\lambda\delta-\mu\delta$
$1-\lambda\delta-\mu\delta$
$\lambda\delta$
$\mu\delta$
0
1
2
$n-1$
n
$n+1$

Figure 5.1: State transition diagram of the discretized $M/M/1$ queue

The probabilities in Equations (5.2), (5.5) and (5.6) add up to 1 plus $o(\delta)$. So only these events where at most one arrival or departure happens are likely to occur and the probability of more than one arrival or departure in the interval I_k is insignificant for small δ. Thus it follows that P_{ii} $(i \geq 1)$ which is the probability of equal number of departures and arrivals in the interval I_k is within $o(\delta)$ of probability in Equation (5.2). The other state transition probabilities $P_{i,i+1}$ and $P_{i,i-1}$ can be argued to be within $o(\delta)$ of the values in Equations (5.5) and (5.6). The state transition diagram of the Markov chain representing the discretized $M/M/1$ queue is shown in Figure 5.1 where the $o(\delta)$ terms have been omitted for clarity.

5.3.2 Stationary Distribution

The stationary distribution of a Markov chain consists of the steady-state probabilities of finding the system in a particular state, i.e p_n, $n = 0, 1, 2, ldots$. For the Markov chain described by the state transition diagram given in Figure 5.1, the stationary distribution is known to exist because the chain is irreducible and all its states are positive recurrent. These properties of a Markov chain are outside the scope of this course. The bottomline is that this Markov chain has a stationary distribution.

The steady-state probabilities can be interpreted as the frequency with which a state is visited by the system and are given by

$$p_n = \lim_{k\to\infty} \Pr\{N_k = n\} = \lim_{k\to\infty} \frac{\sum_{l=0}^{k} \text{In}(N_l = n)}{k}$$

where $\text{In}(\cdot)$ is the indicator function which takes the value 1 if N_l is equal to n and the value 0 otherwise. So the summation in the above equation equals the number of l's for which the system is in state n in the interval $[0, (k+1)\delta]$.

In any time interval, the number of transitions from state n to $n+1$ must differ from the number of transitions from state $n+1$ to n by at most 1. As the time interval of interest goes to infinity, the frequency of transitions state n to $n+1$ is equal to the frequency of transitions from state $n+1$ to state n. Thus the probability that the system is in state n and makes a transition to state $n+1$ is equal to the probability that the system is in state $n+1$ and makes a transition to state n, i.e.

$$p_n\lambda\delta + o(\delta) = p_{n+1}\mu\delta + o(\delta).$$

Dividing both sides of this equation by δ and taking limit as $\delta \to 0$, we get

$$p_n\lambda = p_{n+1}\mu \Rightarrow p_{n+1} = \rho p_n$$

where $\rho = \frac{\lambda}{\mu}$. Consequently, we have

$$p_{n+1} = \rho^{n+1} p_0, \quad n = 0, 1, 2, \ldots \tag{5.7}$$

Since system has to be in some state at any time, we have

$$\sum_{n=0}^{\infty} p_n = 1 \Rightarrow \sum_{n=0}^{\infty} \rho^n p_0 = 1$$

If $\rho < 1$, we get

$$\sum_{n=0}^{\infty} \rho^n p_0 = \frac{p_0}{1-\rho} = 1 \Rightarrow p_0 = 1 - \rho.$$

Substituting the value of p_0 in Equation (5.7), we get

$$p_n = \rho^n (1-\rho), \quad n = 0, 1, 2, \ldots \tag{5.8}$$

Now we can calculate the average number of customers $\bar{N}$ as

$$\bar{N} = \sum_{n=0}^{\infty} n p_n = \sum_{n=0}^{\infty} n \rho^n (1-\rho) = \frac{\rho}{1-\rho} = \frac{\lambda}{\mu - \lambda}$$

As the value of $\rho \to 1$, $\bar{N} \to \infty$.

Then by Little's theorem, the average time spent by the customer in the system is given by

$$T = \frac{\bar{N}}{\lambda} = \frac{\rho}{\lambda(1-\rho)} = \frac{1}{\mu - \lambda}$$

The average time spent waiting in the queue W is the difference between the average system time and the average service time which is given by

$$W = \frac{1}{\mu - \lambda} - \frac{1}{\mu} = \frac{\rho}{\mu - \lambda}.$$

Again by applying Little's theorem, the average number of customers in queue is

$$\bar{N}_q = \lambda W = \frac{\lambda \rho}{\mu - \lambda} = \frac{\rho^2}{1-\rho}$$

5.4 The $M/G/1$ Queueing System

The $M/G/1$ queueing system is a single-server queueing system in which the customer interarrival times are exponentially distributed but the service times have a general distribution. The storage capacity and customer population are assumed to be infinite.

Let X_i be the service time of the ith customer. We assume that the service times $(X_1, X_2, \ldots)$ are independent and identically distributed. They are also assumed to be mutually independent of the interarrival times. Let $\overline{X}$ denote the average service time. Then

$$\overline{X} = E[X_i] = \frac{1}{\mu}$$

where μ is the service rate. Let the second moment of the service time be $\overline{X^2} = E[X_i^2]$.

The average time spent by a customer waiting in the queue is given by the *Pollaczek-Khinchin (P-K) formula*,

$$W = \frac{\lambda \overline{X^2}}{2(1-\rho)}$$

where $\rho = \frac{\lambda}{\mu} = \lambda \overline{X}$. Then the total time spent by a customer in the system on the average is given by

$$T = \overline{X} + \frac{\lambda \overline{X^2}}{2(1-\rho)}$$

By the application of Little's theorem, the average number of customers in the system and the average number of customers waiting in the queue are given by

$$\begin{aligned} \bar{N} &= \lambda T = \rho + \frac{\lambda^2 \overline{X^2}}{2(1-\rho)} \\ \bar{N}_q &= \lambda W = \frac{\lambda^2 \overline{X^2}}{2(1-\rho)} \end{aligned}$$

respectively.

For exponentially distributed service times, $\overline{X^2} = \frac{2}{\mu^2}$ and the average time spent by a customer waiting in queue is given by the same value derived in the previous section for the $M/M/1$ queue.

$$W = \frac{2\lambda}{2\mu^2(1-\rho)} = \frac{\lambda}{\mu(\mu-\lambda)} = \frac{\rho}{\mu-\lambda}$$

Chapter 6

Popular Networking Technologies

6.1 Ethernet (IEEE 802.3)

- Ethernet was invented at Xerox Palo Alto Research Centre (PARC) in the mid-1970s to solve the problem of connecting the computers which Xerox has installed in buildings which were close to each other. Connecting these computers using telephone lines would have been expensive so the Ethernet concept was proposed as an cheaper alternative by Bob Metcalfe.
- 10 Mbps Ethernet standard defined in 1978 by DEC, Intel and Xerox (DIX). This formed the basis for the IEEE 802.3 standard.
- An Ethernet segment is implemented on a coaxial cable which could be upto 500 m in length. The cable is called 10Base5 where the "10" means that the network operates at 10 Mbps, "Base" means that the baseband signaling is used in the cable and "5" means that the segment cannot be longer than 500 m.
- Hosts would connect to the cable using a vampire tap with adjacent taps being at least 2.5 m apart. Multiple Ethernet segments can be joined using digital repeaters upto a maximum of four such repeaters between any pair of hosts. So the total length of an Ethernet-based network can be 2500 m.
- Any signal sent on the Ethernet by a host is broadcast over the entire network by the repeaters. The signaling method used is the Manchester line coding scheme.
- Ethernet can also be implemented on a thinner coaxial cable called 10Base2 where the maximum segment length is 200 m. But the most common wiring technology used in Ethernet is 10BaseT where the "T" stands for the twisted pair of copper wires which constitute the wiring in this case. The maximum length of an Ethernet segment which uses 10BaseT is 100 m.
- The common configuration of 10BaseT-based Ethernet is to have several point-to-point segments coming out of a *hub* which is nothing but a multiway digital repeater.
- An Ethernet frame consists of the following fields
  - A 64-bit preamble which allows the receiver to synchronize to the transmitted signal. It consists of a sequence of alternating 1 and 0 values.

  - A 48-bit destination address and a 48-bit source address.

    These addresses are globally unique across all Ethernet adapters manufactured in the world and are referred to as the *MAC address* of the device. Note that a node can have multiple Ethernet adapters installed on it and each device will have a different MAC address. To make sure that every Ethernet device has a unique MAC address, the first 24 bits correspond to the unique identifier of the manufacturer of the device. The manufacturer is responsible for ensuring that the remaining 24 bits are unique for all the adapters it produces.
  - A 16-bit packet type field which indentifies which network layer protocol the frame should be passed to at the destination.
  - A data field which consists of 46 to 1500 bytes of data.
  - A 32-bit CRC field.

- From the node's perspective, an Ethernet frame consists of a 14-byte header and the data. The Ethernet adapters are responsible for adding the preamble and CRC bits before transmission at the source node and for removing these bits upon reception before passing it on to the destination node.

- The above description is for a DIX Ethernet frame. The 802.3 frame format differs from the DIX frame in two ways.

  - The preamble consists of 56 bits of alternate 1 and 0 values followed by a 8-bit *Start Frame* delimiter having the pattern 10101011. The two consecutive 1 values signal the start of the frame.
  - The 16-bit type field is replaced by a 16-bit length field which gives the length of the frame. IEEE 802.3 uses a portion of the data field to identify the network layer protocol to which the frame should be passed to.

- By the time the IEEE 802.3 standard was published, DIX Ethernet was in widespread use. So interpreting the type field as a length field would have required changes to existing software and hardware. Fortunately, the type fields used by DIX Ethernet for network layer protocols all had values greater than 1500 which is the maximum length of the data field. So a single adapter could accept both types of frame formats if it interpreted the value in the last two bytes of the header as the type of the network layer protocol if it was greater than 1500 and as the length of the frame if it was at most 1500.

- An Ethernet adapter recognizes listens to all the frames being sent on the channel and passes only those frames whose destination address matches its own MAC address to the node it is attached to. Most adapters can also be placed in *promiscuous mode*. In this mode, they pass on all the frames they hear on the channel to the node. If the destination address in a frame is that of a particular Ethernet adapter it is called a *unicast* address. An address which consists of all 1s is called a *broadcast* address. An Ethernet frame which has a broadcast address as the destination address will be passed by an Ethernet adapter to the node even if it is not in promiscuous mode. An address which has the first bit set to 1 but is not a broadcast address is called a *multicast* address. A node can program its Ethernet adapter to accept some set of multicast addresses. Multicast addresses are used to send frames to a subset of nodes in an Ethernet.

- The MAC protocol used in Ethernet in 1-persistent CSMA/CD, i.e. a node listens to the channel before transmitting. If the channel is idle, it transmits immediately. If the channel is busy it waits for the ongoing transmission to complete and then transmits the frame. If it detects a collision, it aborts transmission. We discussed previously that node has to listen for one whole round-trip time before it can decide that its transmission did not suffer a collision. But a node listens to the channel only for the duration of the frame transmission. If the frame is too short then it will stop listening if it does not have other frames to transmit and may miss the occurrence of a collision. For Ethernet, the maximum round-trip time in the presence of four repeaters was calculated to be about 50 μs which us corresponds to 500 bits on a 10 Mbps link. So the minimum length of an Ethernet frame is chosen to be 512 bits to be on the safe side. This is equal to 64 bytes which is why the data field has a minimum length of 46 (the header and CRC make up for the remaining 18 bytes). If a node wants to send less than 46 bytes of data, the remaining bits are padded.

- When a collision occurs, the nodes involved in the collision detect it and implement a backoff algorithm. The time axis is divided into slots of length equal to the maximum round-trip time. Each node involved in the collision first transmits after 0 or 1 slot times where the slot they transmit in is randomly chosen. If a collision occurs again, the nodes involved randomly transmit after 0, 1, 2 or 3 slot times. Thus after the nth collision, the nodes involved randomly wait for 0, 1, 2, . . . , $2^n - 1$ slots before transmit. After ten collisions, the randomization interval is fixed at a maximum of 1023 slots. After 16 collisions, the data link layer gives up and reports an error to the higher layers. This algorithm is called *binary exponential backoff*. It can be thought of as an adaptive collision resolution algorithm which adapts to the number of nodes involved in the collision.
- If a CRC error is detected, the frame is discarded by the destination node. Ethernet does not use a ARQ protocol. The higher layer protocols are responsible for resending erroneous frames.

6.2 IEEE 802.11

- The IEEE 802.11 family of protocols are the most popular wireless LAN protocols in use today. They can operated in two modes - without a base station and with a base station.
- IEEE 802.11 supports six different kinds of physical layers - five of them are based on spread spectrum modulation and one if based on infrared signals.
- The original 802.11 standard operated at 2 Mbps and defined two physical layers - one based on frequency hopping spread spectrum (FHSS) and the other based on direct sequence spread spectrum (DSSS). The next variant which was developed was 802.11b which operated at 11 Mbps in the 2.4 GHz unlicensed band and used high rate DSSS.
- The first high-speed wireless LAN standard was IEEE 802.11a which used OFDM signaling and supported upto 54 Mbps data rate in the unlicensed 5 GHz band. The 5 GHz band suffers from high signal absorption and so the IEEE 802.11g standard was developed to operate at 54 Mbps in the 2.4 GHz band.
- IEEE 802.11 supports two modes of MAC layer operation. The first one is called the *Distributed Coordination Function (DCF)* and does not require any centralized control. The other one is called the *Point Coordination Function (PCF)* and requires a base station to coordinate all the transmissions among the nodes. All implementations of 802.11 must support DCF but support for PCF is optional.
- In DCF, the MAC protocol used is CSMA/CA, i.e. carrier sensing along with a RTS-CTS mechanism to mitigate the hidden node and exposed node problems.

- As probability of frame error increases with the size of a frame, a frame can be fragmented and multiple fragments can be sent after an RTS-CTS exchange.
- In PCF, the base station polls each node asking it if it has any frames to send. Thus the transmissions from the nodes are completely controlled by the base station avoiding the possibility of collisions.
- The ARQ protocol used is stop-and-wait ARQ.

Chapter 7

Network Layer

While the data link layer was concerned providing error-free communication between adjacent nodes, the network layer is the lowest layer which is concerned with end-to-end communication between possibly non-adjacent source and destination nodes. The main issues handled by the network layer are the following.

- *Internetworking:* The network layer enables internetworking, i.e. the seamless interconnection of heterogeneous networks. This makes communication between nodes connected to different networks possible. The most successful network layer protocol is the Internet Protocol (IP) which enables internetworking on the Internet by the encapsulation of IP packets within the frames of different networks.
- *Routing:* The network layer handles the tasks of discovering routes from source to destination nodes and selecting the best routes among multiple alternatives.
- *Congestion avoidance:* The network layer is also responsible for preventing congestion in the network, i.e. buffer overflows occurring anywhere in the communication network. This is different from flow control where the goal is to prevent a single source from overflowing the buffer at a particular destination.

As discussed in Section 2.3, the Internet Protocol (IP) which is the mainstay of the TCP/IP protocol suite will be used to illustrate the functionalities of the network layer. This is in contrast to our coverage of the physical and data link layers where our discussion was more generic.

7.1 Internetworking

An *internetwork* or *internet* (note the lowercase *i*) is a collection of networks which have been interconnected to provide communication between any two arbitrary nodes in any of the constituent networks. In this context, a *network* is defined to be a directly connected collection of nodes which use the same physical and data link layer technologies for communication between them. The networks constituting an internet are hetergeneous, i.e. they use different physical layer signaling formats, different data link layer frame structure, addresses etc. For example, an Ethernet-based network could be connected to a 802.11 network to form an internet. Ethernet segments connected by repeaters are not considered to form an internet. Another example is a company connecting the Ethernet networks existing in their branch offices in Delhi and Mumbai with a point-to-point link leased from a telephone company. Although, in this case the networks being connected employ the same technology (Ethernet) this is considered an internet because the point-to-point fiber optic link uses a different physical and data link layer technology. The nodes which interconnect the networks in an internet are called *routers*. Routers need to have an network adapter for each kind of network they are connected to.

The Internet Protocol (IP) is the most successful technology today for building scalable, hetergeneous internetworks. The key idea is to assign an *IP address* to all the nodes in an internetwork and embed IP packets within the frames used by the different constituent networks. Consider the example of a source node on an Ethernet network A wanting to communicate with a destination node on an Ethernet network B when both the networks are connected by a point-to-point fiber optic link. The source node encapsulates its payload in an IP packet and embeds the IP address of the destination node as the destination address in the header of the IP packet. This source node passes this IP packet to its Ethernet adapter. The Ethernet adapter encapsulates the IP packet in an Ethernet frame and sends it to the Ethernet adapter of the router which acts as the interface between the network A and fiber optic link. The router extracts the IP packet from the received Ethernet frame and looks at the destination IP address. Upon finding the destination IP address to be equal to that of a node reachable through its fiber optic link, it encapsulates the IP packet into a frame corresponding to the fiber optic technology and sends it to the router on the other side of the fiber optic link which is directly connected to network B. The router connected to network B extracts the IP packet from the received frame. Upon finding the destination IP address to be that of a node reachable through its Ethernet interface, it embeds the IP packet in an Ethernet frame with the destination MAC address equal to the MAC address of the destination node and transmits it on network B.

This method of forwarding packets follows the *datagram* or *connectionless* model of data delivery where every packet contains sufficient information to allow the network to forward it to the correct destination. This is in contrast to a *connection-oriented* model where an advanced connection setup between the source and destination is required before packets can be forwarded. The advantage of the latter is that packets will arrive in the order they were sent and each packet needs to have only the identity of the connection it belongs to, which can be a stored in a much smaller field than the destination address. Of course, the overhead and delay involved in connection setup is a disadvantage of the connection-oriented model. The data delivery model of IP is a *best effort* or *unreliable* service because it does not react to failures in delivery like packet loss, corruption and misdelivery. Delivery failure also includes situations when packets are delivered out of order or more than once. The responsibility of handling such failures rests on the higher-layer protocols or applications.

IPv4 Packet Structure

The current version of IP is 4 and it called IPv4. The next generation of IP has version number 6 and is called IPv6. It is discussed later in this chapter. The IPv4 packet structure is shown in Figure 7.1 where each row corresponds to 32 bits or one word. The numbers above the first first row indicate the bit offset of the fields beginning at that position. The packet consists of the following fields.

- **Version** The first header field in an IP packet is the 4-bit version field. For IPv4, this field has the value 0100 corresponding to the integer value 4. Having the version field right at the beginning makes the implementation of the packet processing system simpler. The

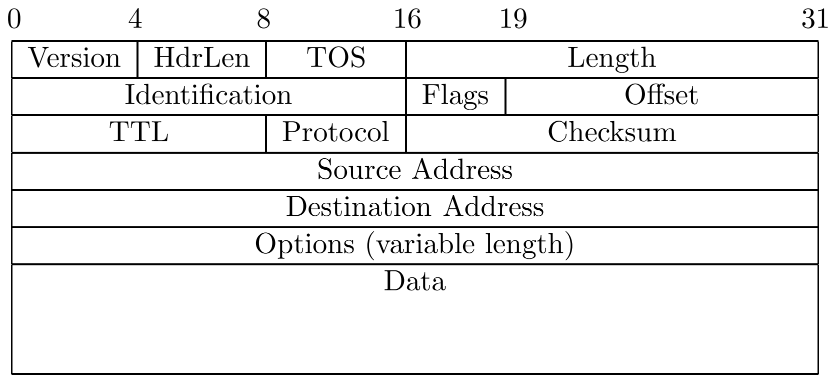


Figure 7.1: IPv4 packet structure

processing of the packet can be different depending on the version.

- **Header Length** The second field is the 4-bit header length field which specifies the length of the header in 32-bit words. The minimum value of this field is 5 which corresponds to a 160-bit header (5×32).
- **TOS** The 8-bit *Type of Service (TOS)* field was originally intended to be used by a source of the packet to specify how it should be handled at a router. Now the meaning of the bits in this field has changed and the field is referred to as the *Differentiated Services (DiffServ)* field. The first two bits are reserved for use by the *Explicit Congestion Notification (ECN)* protocol and the last six bits are reserved for use by the DiffServ protocol.
- **Length** The next 16 bits of the header contain the length of the IPv4 packet in bytes of the header and the payload. Thus the maximum length of the packet can be 65,535 bytes.
- **Identification** The first 16 bits of the second word of the header contain an identification field which is used for identifying the fragments of an IP packet. Fragmentation is discussed in detail in a later section.
- **Flags** The next 3 bits contain flags which are used to implement the fragmentation algorithm.
- **Offset** The remaining 13 bits which make up the second word of the header contain the offset of a particular fragment relative to the beginning of the original unfragmented packet measured in eight-byte blocks. So the maximum offset can be $(2^{13} - 1) \times 8 = 65,528$.
- **TTL** The first 8 bits of the third word of the IPv4 header contain the *Time To Live (TTL)* field. This field is decremented each time a router receives the packet and the packet is dropped once the field becomes zero. This is to prevent IP packets from being routed indefinitely in an internet. Historically this field was set to a certain number of seconds that a packet would be allowed to live. However, since routers do not have access to a common clock they just decrement the field by 1 before forwarding the packet. So this field has in effect become a hop count field. The default value of this field is 64.
- **Protocol** This 8-bit field specifies the transport layer protocol to which the IP packet should be passed to. This field has value 6 for TCP and value 17 for UDP.
- **Checksum** The 16-bit checksum is calculated by grouping the header bits into 16-bit words, taking their 1's complement sum and taking the 1's complement of the result. A

odd number of bit errors in any of the 16 positions can be detected using this checksum. This checksum does not have the same error detection capabilities as a CRC but it has an easy software implementation.

- **Source Address** The next header field contains the 32-bit IPv4 address of the source node.
- **Destination Address** The next header field contains the 32-bit IPv4 address of the destination node. This is the last field which is required to be present in any IPv4 packet header.
- **Options** Additional optional header fields may be present in the IPv4 header. These are usually not present and are used only for debugging and measurement purposes. Details can be found in RFC 791 `http://www.rfc-editor.org/rfc/rfc791.txt`. Also see `http://en.wikipedia.org/wiki/Request_for_Comments` to know more about RFCs in general.

Internet Checksum

The checksum used in the IPv4 header is called the *Internet checksum*. The same algorithm is also used in TCP and UDP. In this algorithm, the bits to be checksummed are grouped into 16-bit words and their 1's complement sum is calculated. The 1's complement of this sum then gives the checksum. For example, suppose we want to calculate the Internet checksum of the following 32 bits : 11111110 11001000 00000010 00010011. We add the first 16 bits to the second 16 bits using 1's complement arithmetic where it is necessary to add the carry back to the sum of the bits. The checksum is then given by the 1's complement of the sum as 11111111 00100011.

	11111110	11001000	
	00000010	00010011	
1	00000000	11011011	
		1	Adding carry bit
	00000000	11011100	1's complement sum

Since the checksum field is part of the IP header, it is set to all zeros for the calculation of the Internet checksum. It is then filled with the resultant checksum. To verify the checksum at a receiver, the 1's complement sum of the IP header is recalculated. If it equals all 1's, then no errors have been detected.

IP Fragmentation and Reassembly

Different network technologies specify a different maximum payload length they can support in a single frame. This specifies the *maximum transmission unit (MTU)* of the network which is the largest IP datagram which the network can carry in a frame. For example, the MTU in Ethernet is 1500 bytes long while in 802.11 it is 2304 bytes long. Since an internet can consist of networks which support different MTUs, there can be two approaches to enabling internetworking using IP. The first approach is to choose the size of the IP datagrams to be small enough to fit in one frame of any of the network technologies used in the internet. The second approach is to allow fragmentation of IP datagrams when they cannot fit in the frame of a particular network technology. The latter approach seems to be more efficient as the source does not needlessly send small IP datagrams which increase the amount of overhead sent as headers. The fragments of an IP datagram are reassembled at the destination.

Fragmentation is typically performed by a router when it receives an IP datagram which needs to be forwarded over a network whose MTU size is smaller than the size of the datagram. When the datagram is broken into fragments, the IP header of the datagram is added to all the fragments. The identification field in the original datagram is chosen by the source node and is used at the destination to identify all the fragments of a datagram. The 3-bit flags field plays a special role in the fragmentation process. One of the bits is reserved and always set to 0. One of the other two bits is the *More Fragments (MF)* flag which is set to 1 in all the fragments except the last one. It is also set to 0 in unfragmented IP datagrams. The third flag bit is called the *Don't Fragment (DF)* flag which is set to 1 by the source if it does not want the datagram to be fragmented. This is used to send packets to a destination which does not have the capability or resources to handle the reassembly process. If a datagram with the DF flag set to 1 is received at a router and fragmentation is required to send it across the next hop, then the datagram is dropped.

Fragmentation occurs at 8-byte boundaries and the offset field in the header of a fragment specifies the offset of the current fragment relative to the beginning of the original datagram. For example, suppose a 1420 byte datagram which has 1400 bytes of data and 20 bytes of header arrives at a router and needs to be sent over a network which has an MTU of 532 bytes. An MTU size of 532 bytes allows for a 512 byte data portion after 20 bytes are taken up by the IP header. So the first fragment contains the first 512 bytes of the 1400 bytes of data in the original datagram. The header of the first fragment is identical to the header of the original datagram except that the MF flag is set to 1. The offset field in the original datagram is zero since it is unfragmented and the offset field in the first fragment is also zero. The second fragment also contains 512 bytes of data which corresponds to bytes 513 to 1024 in the original 1400 bytes of data. The header of the second fragment is the same as that of the first fragment except that the offset field is set to 64 which is 512 divided by 8. The third fragment has the remaining 376 bytes of data and the header is identical to that of the other fragments except that the MF flag is set to 0 and the offset field is set to 128 which 1024 divided by 8.

The fragment reassembly process can consume significant memory resources at the destination if some of the fragments are lost. This is because the destination has to reserve memory for the fragments which have arrived while waiting for the lost fragments to be retransmitted. This phenomenon has been the basis of a denial-of-service attack where an attacker sends several IP datagrams with the MF flag set to 1. The receiving nodes keeps allocating memory to store these datagrams while it waits for the fragment with the MF bit set to 0 to arrive eventually resulting in a buffer overflow. For this reason, IP fragmentation is generally avoided. An alternative to fragmentation is to have the source node perform *path MTU discovery* where the smallest MTU in the path from source to destination is calculated. In this protocol, the source node sends an echo packet to the destination where the size of the echo packet is equal to the MTU of the network the source is directly connected to. This packet also has the DF flag set to 1 in its header. If the echo packet returns successfully, the MTU size is small enough to reach the destination without fragmentation. If the echo packets encounters a network with a smaller MTU size, the router attached to the network sends an error message back to the source. The source then reduces the size of its echo packet and tries again.

IP Address Notation

The IP address is a network layer address which has no relation to the MAC addresses used in the data link layer. It is 32 bits long in IPv4 and 128 bits long in IPv6.

IPv4 addresses are written in *dotted decimal* notation. In this notation, each 8-bit byte in the 32-bit address is written as an integer between 0 and 255. For example, the IPv4 address 0000 1010 0110 1011 0000 0001 0000 0001 written as 10.107.1.1.

IPv6 addresses are written in hexadecimal notation where the 128-bit address is written as 32 hexadecimal digits where groups of four hexadecimal digits are separated by colons. Two examples of IPv6 addresses are the following.

```
FE80:AB89:200A:ABCD:EFAB:1234:5678:9ABC
FE80:0000:0000:0000:0000:0250:56FF:FEC0
```

Since these addresses are long, a common way to abbreviate them is for a group of four hexadecimal digits to leave out leading zeros and for groups containing four zeros to be represented by a single zero. As long as there is no ambiguity, a long string of zeros in the IPv6 address can be replaced by a double colon (::). The first example of an IPv6 address above cannot be compressed but the second example can be written as

```
FE80::250:56FF:FEC0.
```

A double colon can appear only once in an IPv6 address. To see what can go wrong, consider the IPv6 address

```
FE80:0000:0000:56FF:0000:0000:0000:FEC0.
```

If we compress this address as `FE80::56FF::FEC0` using two double colons, we cannot uncompress it unambiguously because we do not know how many of the 5 groups of missing zeros need to be inserted to the left of `56FF` and how many to the right of `56FF`. The correct way to compress this address is to write it as `FE80:0:0:56FF::FEC0` or `FE80::56FF:0:0:0:FEC0`.

Network and Host Portions of IP Addresses

While the MAC address is assigned to a network adapter by the manufacturer and cannot be changed, the IP address can be configured by the system administrator or user. While the only restriction on MAC addresses was that they be unique, IP addresses are required to have a hierarchical structure in addition to being unique. Each IP address assigned to a node[1] has two parts: a network part and a host part. The network part is the same for all the nodes connected to a single physical network and is used to identify the network. The host part is used to identify the node in the network. This hierarchical structure helps reduce the size of the routing tables and also makes packet forwarding easier. For example, two nodes connected to the same physical network can have IP addresses 10.107.1.1 and 10.107.1.2 where the first three bytes form network portion of their IP addresses (10.107.1) and the last byte forms the host portion of the IP address.

When an IP packet is received at a router it checks if the network portion of the destination IP address in the packet matches the network portion of any of the IP addresses assigned to each

[1]Note that the IP address is assigned to a particular network adapter on the node since the node can have multiple network adapters

of its network interfaces. If a match is found, then the destination node is directly connected to one of the interfaces of the router. The router then sends the packet directly to the destination using that interface. If a match is not found, then the destination node is not present on any of the networks the router is directly connected to. Then the router looks in its routing table to find the *next hop* router for the particular network portion of the destination address in the packet. If the routing table does not contain an entry corresponding to the network portion in the packet, then the packet is forwarded to a *default router* which then assumes the responsibility of delivering the packet to the destination. This method of matching only the network portion of the destination address has the advantage that the routing tables only have to contain entries corresponding to networks rather than all the nodes in the internetwork.

Address Resolution Protocol (ARP)

In the previous subsection, we said that a router sends an IP packet directly to the destination node if it finds that the network portion of the destination IP address matches the network portion of the IP addresses assigned to one of its interfaces. But to send a packet to a particular destination node, the router needs to know the MAC address, i.e. the data link layer address of that node so that the packet can be embedded in a frame which has that particular MAC address as the destination address. Even when the packet has to be sent to a next hop router or default router, the MAC addresses of these routers are needed. So each node has to maintain a table of mappings from IP addresses to MAC addresses. While this table can be loaded into the nodes by the system administrator, a better approach is to have the nodes dynamically learn the entries in the table. This is done using the *Address Resolution Protocol (ARP)*. The set of mappings stored in a node is called the *ARP cache* or *ARP table*. Since the mappings can change over time, each entry in the ARP cache has a timer associated with it. When ARP entries have not been updated for a while, their timers expire and they are removed.

ARP takes advantage of the fact that many data link layer technologies support broadcast. When a source node wants to send an IP packet to a destination it knows is connected to the same network it is connected to, it first checks the ARP cache for a mapping. If the mapping is not present, it broadcasts an ARP query into the network. In Ethernet, this is done by setting the destination MAC address to be all 1s. The ARP query has the IP address of the intended destination. Each node in the network receives the ARP query and checks if the IP addresss in the query matches its own IP address. If there is a match, the destination sends a response containing its MAC address back to the source node. The source node maps the received MAC address to the IP address and adds this mapping to the ARP cache. The ARP query contains the IP address and MAC address of the source node. So every node which receives the query can add this mapping to its own ARP cache. But this is not what happens. If a node already has a mapping for the source node, it refreshes the mapping, i.e. it replaces the mapping if it has changed and in the case of no change it resets the timer which will remove the mapping on expiry. If a node does not have a mapping for the source node, it adds the mapping only if it is the target of the ARP query. This is because it is highly likely that the destination node would have to send a response or an acknowledgement back to the source node and will need the source's MAC address. If a node is not the target of the ARP query it does not add a mapping for the source node to its ARP cache because there is not enough reason to believe that it will have to send a packet to the source node in the near future and adding the mapping will unnecessarily clutter the ARP cache.

Class	Initial bits	Num. of addresses	% of address space	NetID	HostID
Class A	0	2^{31}	50%	8 bits	24 bits
Class B	10	2^{30}	25%	16 bits	16 bits
Class C	110	2^{29}	12.5%	24 bits	8 bits
Class D	1110	2^{28}	6.25%	-	-
Class E	1111	2^{28}	6.25%	-	-

Table 7.1: IPv4 classful addressing scheme

IPv4 Addressing

The original IPv4 addressing scheme was called the *classful* scheme because it divided the 32-bit address space into classes based on the value of the initial bits in the IPv4 address. The first three classes also defined the number of address bits which are assigned to the network and the host portion of the IP address. This is shown in Table 7.1. Class A addresses have their first bit set to 0, class B addresses have their first two bits set to 10, class C addresses have their first three bits set to 110, class D addresses have their first four bits set to 1110 and class E addresses have their first four bits set to 1111. Class A, B and C addresses can be assigned to nodes while class D addresses are assigned to multicast groups and class E addresses are reserved for experimental purposes and seldom used. Class A networks have 7 bits for the network part (the first bit in the 8-bit network part is set to 0) and 24 bits for the host part. So there can be $2^7 = 128$ class A networks. But there are actually 126 such networks because the values 0 and 127 in the network part of the address are reserved. Each class A network can have $2^{24} - 2$ hosts (again the all 0s and all 1s host parts are reserved). Similarly each of the 2^{14} class B networks can have 65,534 hosts and each of the 2^{21} class C networks can have 254 hosts.

The problem with the classful addressing scheme was that each network part was supposed to identify exactly one physical network. This meant that the address space would be severely underutilized if the number of hosts in a physical network did not match one of the three classes available. For example, a physical network with two nodes would require a whole class C network resulting in the other 252 IP addresses remaining unused. The underutilization was more severe if a physical network had slightly more than 254 nodes because in that case the address space of a class B network would be required and more than 65,000 IP addresses would remain unused.

Today IPv4 addresses are interpreted in a *classless* manner. In classless addresssing, the boundary between the network and host portions of the IPv4 address is determined by the IPv4 *network mask* or *network prefix* and not by the initial IP address bits. The network mask or prefix is a string of bits as long as the IP address. For IPv4, the network mask is 32 bits long. The mask consists of a sequence of 1s followed by certain number of 0s. The bit positions which are set to 1 determine the network portion of the IP address. The network mask can also be written in dotted decimal notation for IPv4 addresses. For example an IPv4 address 10.107.1.1 with network prefix 255.255.255.0 tells us that the network portion of the address consists of the first three bytes (24 bits) 10.107.1 and the hosts on this network can have addresses in the range 10.107.1.1 to 10.107.1.254. The address 10.107.1.0 is used to represent the network and the address 10.107.1.255 is used as a network layer broadcast address. The classes A, B, C had networks masks 255.0.0.0, 255.255.0.0 and 255.255.255.0 respectively. Since the numbers of 1s in the network mask uniquely defines it, a network IP address with a mask can be written in the format *network id/prefix length*. In our above example, we can write the network address along with the three byte prefix as 10.107.1/24 or 10.107.1.0/24.

The classless method of interpreting IP addresses allows the use of two techniques which result in better utilization of the IP address space and also improve routing scalability: *subnetting* and *supernetting*. Subnetting involves sharing the host IP addresses corresponding to a single network address among multiple physical networks. These physical networks are called the *subnets* of the original network. The key idea is to configure all the hosts in a subnet with a network mask which in this context is also called the *subnet mask*. While the restriction in classful addressing was that all the nodes in the same physical network need to have the same network address, in classless addressing with subnetting all the nodes in the same physical network need to have the same subnetwork address which is the bitwise AND of the node IP address and the subnet mask. For example, suppose we have two physical networks with 100 nodes each. With the classful addressing scheme, we would have to allocate two different class C network addresses to each of the networks. But with subnetting we can use a single class C network address with a subnet mask of 255.255.255.128. Suppose the class C network address is 192.83.12.0. The hosts in such a network can have IP addresses in the range 192.83.12.1 to 192.83.12.254. Then one of the physical networks can have the subnetwork address 192.83.12.128 and have host IP addresses in the range 192.83.12.129 to 192.83.12.254. The other physical network can have subnetwork address 192.83.12.0 and have hosts IP addresses in the range 192.83.12.1 to 192.83.12.126. While this solves the problem of address space underutilization, it helps routing scalability if the subnetworks are close to each other. If they are close, they could both interface to the outside world through a single router which can advertise a single network address for both subnetworks. In our example, the network address would be 192.83.12.0 with network mask 255.255.255.0. This would reduce the routing table size since only one entry needs to be stored for both the subnetworks.

The other technique is *supernetting* which is also called *classless interdomain routing (CIDR)*. Consider a scenario where a university campus network has 16 class C network addresses assigned to it. Assigning a class B network address to this network will lead to underutilization of the available host addresses. But having 16 class C network addresses creates the problem of maintaining 16 entries in the routing tables of the routers in the Internet to reach the nodes on the campus network even if all the routes are the same. The solution is to allocate a contiguous block of 16 class C network addresses and use a network mask of 20 bits to identify the campus network as a single network to the routers on the Internet. For example, if the class C network addresses allocated are 192.64.16 to 192.64.31 then the first 20 bits of the addresses in this range are the same (11000000 01000000 0001). So this group of 16 class C networks can be represented by a single network with network address 192.64.16/20. This will result in an aggregation of the 16 routing table entries into a single entry.

Public and Private IP Addresses

A public IP address is one which is globally unique on the Internet and is assigned to a node by a central authority or by an Internet Service Provider (ISP) who in turn got it from a central authority. The Internet Corporation for Assigned Names and Numbers (ICANN) oversees the allocation of public addresses. The actual allocation is done by Regional Internet Registries (RIRs) like the American Registry for Internet Numbers (ARIN), Reseaux IP European Network Coordination Center (RIPE NCC), Asian Pacific Network Information Center (APNIC), Latin American and Caribbean Network Information Center (LACNIC) and African Network Information Center (AfriNIC).

Private IP addresses are meant to be used only on local networks and are not guaranteed to be globally unique. For this reason, private IP addresses should not be seen in packets outside a local network. Consequently, private addresses must be mapped to public addresses whenever a packet with a private source IP address leaves the local network and enters the Internet. Despite this restriction, private IP addresses are used frequently because they prevent public IPv4 address exhaustion. Private address spaces for classes A, B and C were defined in RFC 1918 and they are valid under the classles addressing scheme.

- *Class A*: 10.0.0.0 through 10.255.255.255 (10.0.0.0/8)
- *Class B*: 172.16.0.0 through 172.31.255.255 (172.16.0.0/12)
- *Class C*: 192.168.0.0 through 192.168.255.255 (192.168.0.0/16)

Network Address Translation (NAT)

While IPv6 is a long-term solution to the problem of IPv4 address space exhaustion, *network address translation (NAT)* emerged as a technology which conserved the IPv4 address space. The main idea behind NAT is to use private IPv4 addresses for hosts in a local network and translate these addresses to public IPv4 addresses only when the hosts need to communicate with external hosts located outside the local network. This translation is done by a NAT server which is a router acting as the interface between the local network and the outside world. All traffic between local hosts and hosts outside the local network has to pass through the NAT server. The main function of the NAT server is to translate the private IPv4 addresses in the outgoing packets to globally unique public IPv4 addresses and the globally unique IPv4 addresses in the incoming packets to private IPv4 addresses which identify hosts on the local network.

There are several types of NAT implementations but the two which have resulted in the conservation of the IPv4 address space are *dynamic NAT* and *port-based NAT*. In dynamic NAT, a small set of public IPv4 addresses are shared by a larger set of hosts for communicating with the outside world. For example, consider a local network consisting of 200 hosts that uses private addresses in the range 10.107.1.0/24. Suppose these hosts use dynamic NAT to share a pool of 20 public IPv4 addresses in the range 192.83.12.1 to 192.83.12.20. When a host with IP address 10.107.1.1 wants to communicate with a website with IP address 250.110.11.5 located outside the local network, it sends an IP packet with source address 10.107.1.1 and destination address 250.110.11.5 to the NAT server. The NAT server replaces the source IP address with a public IP address, say 192.83.12.7, from the pool of 20 IP addresses, recalculates the IP checksum and forwards the modified IP packet to the website. When the NAT server receives a response IP packet from the website with destination IP address 192.83.12.7, it replaces the destination IP address with the IP address 10.107.1.1 and forwards it to the original source host on the local network. The mapping of local host IP addresses to global IP addresses in the shared pool is not fixed and changes according to which IP addresses are available at a particular time. Dynamic NAT is based on the observation that all hosts on a local network may not want to communicate with external hosts at the same time. If this observation does not hold, the alternative is to use port-based NAT. Port-based NAT is based on the fact that most transport layer protocols like TCP and UDP use a *port number* to identify a process running on a host. This port number is used to demultiplex packets which are received at a destination host to different processes which are running on the host. For example, a host might be running a web browser and an FTP client at the same time which are both receiving packets. A port number in the transport layer header distinguishes the packets which need to be passed to the web browser and the FTP client. The port number is a 16-bit field and hence can take values from 0 to 65,535. In port-based NAT, the NAT server replaces the source IP address and port number with a public

IP address and an arbitrary port number which is unique to the host on the local network in all the outgoing packets. When an incoming packet has the public destination IP address and port number corresponding to a local network host the NAT server it replaces these fields with the private IP address and original port number. For example, consider a local network of 200 hosts that uses private addresses in the range 10.107.1.0/24. Suppose these hosts use port-based NAT to share a single IP address 192.83.12.1. Suppose an FTP client on a local host with IP address 10.107.1.1 wants to send communicate with a website with IP address 250.110.11.5. The local host sends a packet with source IP address 10.107.1.1 and source port number 21 (corresponding to FTP) to the NAT server. The NAT server replaces the source IP address with IP address 250.110.11.5 and source port with an arbitrary port number, say 12345, which it will reserve for this particular local host. It will have to recalculate the IP/TCP checksums before sending the IP packet. When the response IP packet from the website arrives at the NAT server with destination IP address 250.110.11.5 and destination port 12345, the NAT server replaces the destination IP address with the value 10.107.1.1 and destination port with the value 21 and passes the packet to the local host 10.107.1.1. In this way, dynamic NAT and port-based NAT enable the sharing of a small number of public IP addresses by a large number of hosts.

Dynamic Host Configuration Protocol (DHCP)

All the hosts in an internetwork need to be assigned a unique IP address with the restriction that the network portion of the IP address is the same for all hosts connected to the same physical network. One way to configure the IP addresses on hosts is to have a system administrator manually configure them. But this is a time-consuming task especially because the hosts are not reachable through the network before they have been configured. Manual configuration is also prone to errors like two hosts being configured with the same IP address because it is very easy for humans to confuse two IP addresses. In addition, to the IP address other parameters like the subnet mask and default router's IP address need to be configured in the host before it can commence communication.

DHCP is a method to perform automatic initial configuration of the hosts in a network. It is implemented using a *DHCP server* that is responsible for providing configuration information to hosts. Whenever a host boots up or gets connected to the network it retrieves configuration information from the DHCP server. In the case of IP addresses, one way is to have a mapping from IP addresses to host MAC addresses in the DHCP server so that when a host requests the server for its IP address its MAC address is used by the server to find the appropriate IP address. But in this method the system administrator has to manually create the table of mappings. A simpler method to configure IP addresses is to have a pool of IP addresses stored in the DHCP server which are assigned to hosts on demand. In this method, the exact IP address which will be assigned to a host is not known beforehand. However, in both methods the host has to know the IP address of the DHCP server before it can send a request. If each host has to be manually configured with the IP address of the DHCP server, this would defeat the purpose of automatic configuration. Sending a broadcast data link layer frame will not be sufficient because the DHCP server may not be located on the same physical network. The solution is to have a newly booted or connected host to send a *DHCPDISCOVER* message to the special IP address 255.255.255.255 which is an IP broadcast address. This message is embedded in a data link layer broadcast frame. An IP packet with destination address equal to the IP broadcast address will be received by all the hosts and routers in a network. If the DHCP server is not directly connected to the network, one of the routers in the network is configured to be a *relay agent*. The relay agent is configured with the IP address of the DHCP server. When the relay agent receives a DHCPDISCOVER message it unicasts it to the DHCP server and forwards the server's response to the requesting host. The server's response contains the requesting host's MAC address, the IP address which the server is offering, the subnet mask, the lease duration of the IP address and the IP address of the DHCP server. The purpose of the lease is to make sure that hosts do not occupy IP addresses indefinitely even when they do not need them. The hosts cannot be given the responsibility of giving back the IP addresses assigned to them to the DHCP server because they might crash, be disconected from the network or powered off. So by imposing a lease on an IP address, the server is free reclaim the address once the lease expires. A host will have to renew its lease periodically if it wants to continue using the IP address.

7.2 Routing

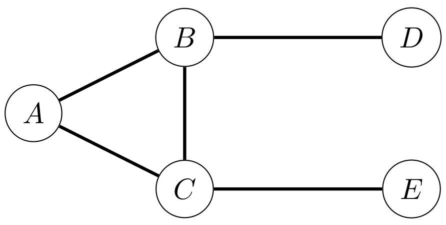
As the size of a network grows, many pairs of nodes will not be directly connected to each other and will depend on intermediate nodes to relay packets between them. A sequence of relay nodes which transfer information between a source-destination node pair is called a *route*. Each node in the network has a *routing table* which consists of a list of mappings from IP addresses to *next hop* nodes. The next hop node corresponding to an IP address is the node to which a packet destined for that address should be forwarded to. To reduce the size of the routing tables, whenever possible the mapping is from network IP address to a next hop node, i.e. if all the nodes with the same network portion of the IP address are reachable through the same next hop node then only one mapping is added in the routing table for all of them. Every node also has a *forwarding table* which consists of a list of mappings from IP addresses to outgoing network interfaces along with the MAC address of the next hop node.

Routing is the process of constructing the routing tables in a network such that packets exchanged between pairs of nodes take paths which have lowest cost among all available paths. Routing is accomplished by a *routing algorithm* which is typically a distributed algorithm involving all the nodes in the network. The main reason for the distributed nature of most routing algorithms is that centralized routing algorithms are not scalable. There are two main classes of routing protocols: *distance vector* and *link state* routing protocols. These protocols are easily illustrated using the graphical model of a network where nodes are represented by vertices on a graph and communication links are represented by edges in the graph. Each edge has a cost associated with it and the goal of the routing algorithm is to find the lowest cost path between any pair of nodes.

Routing in internetworks differs from routing in the graphical model in the fact that the routers participating in the routing algorithm advertise costs of reaching networks rather than hosts. This has the advantage of reducing the number of entries in the routing table.

Distance Vector Routing Protocol (DVRP)

The main idea behind DVRP is that each node constructs a vector containing the shortest distances from itself to all the nodes it knows are reachable from it and distributes this vector to its immediate neighboring nodes. The assumption is that a node knows which nodes are its immediate neighbors and also the costs to these neighboring nodes. The neighboring nodes upon receiving the distance vector update their own distance vectors if they find shorter routes to nodes which are reachable from them or if they find that new nodes which are not present in their distance vector.



(a) A five-node communication network

A's routing table		
RN	NH	RC
B	*B*	1
C	*C*	1

A's routing table		
RN	NH	RC
B	*B*	1
C	*C*	1
D	*B*	2

A's routing table		
RN	NH	RC
B	*B*	1
C	*C*	1
D	*B*	2
E	*C*	2

(b) Illustration of the evolution of the routing table of node A where RN, NH and RC are abbreviations of reachable node, next hop and routing cost, respectively. The routing cost is the hop count here. The leftmost table is A's initial routing table, the middle table is A's routing table after it receives B's distance vector and the rightmost table is A's routing table after it receives C's distance vector.

Figure 7.2: Illustration of distance vector routing

For example, consider the five-node network shown in Figure 7.2a. Suppose each edge has unit cost which corresponds to the hop count being chosen as the routing metric. Initially node A's routing table only has the entries corresponding to its immediate neighbors B and C. Suppose it receives B's distance vector which contains a path to node D at cost 1. Then it adds an entry for node D with next hop B and cost 2 since it can reach node B with cost 1. Next node A receives C's distance vector and adds an entry corresponding to node E with next hop C and cost 2 since it can reach node E with cost 1. A similar process happens at all the nodes which send their distance vectors and update their routing tables upon receiving the distance vectors from their neighbors. If the routing tables at all the nodes stabilize and contain a consistent view of the network, the routing algorithm is said to have *converged*.

Each node sends its distance vector to its neighbors periodically resulting in a *periodic update* even when there are no changes in its routing table. When a change occurs in a node's routing table it immediately sends its new distance vector resulting in a *triggered update*. The periodic update is useful in identifying a node failure which can be characterized by the absence of the periodic update message in the last few update cycles. This routing protocol has the ability to recover from some node or link failures. Suppose the link from B to C in goes down in the network shown in Figure 7.2a. Then B advertises a distance of infinity to C and node D sets its distance to C to infinity because it knows that its path to C is through B. However, A advertises a unit distance to C and B updates its routing table to have a path to C through A having cost 2. Eventually D also updates its routing table to have a path to C through B having cost 3. Thus the routing algorithm converges.

However, some link failures can prevent the distance vector routing algorithm from converging. For example, consider the case when the link from C to E fails or equivalently the case when node E fails. Before the failure, nodes A, B and C have routing costs 2, 2 and 1 to node E respectively. After the failure, node C sets its routing cost to node E to infinity. But suppose node A advertises a cost of 2 to reach E before getting the distance vector from C. Then node

B believes that it can reach E through A at a cost of 3. It then advertises this to C who then believes that it can reach E through B at a cost of 4. When C advertises this to its neighbors, A updates its cost to reach E to 5. Then B updates its cost to reach E to 6 and so on. This cycle of updates stops only when the costs become large enough to be considered infinite. This is called the *count to infinity* problem.

One solution to the count to infinity problem is to set a relatively small cost as infinity. For example, if we set 16 to be infinity then the nodes will count to 16 fairly quickly and realize that there is no path to the node which failed. This will work only if the maximum number of hops in a network is 16. Another technique to solve this problem is called the *split horizon.* In this technique, when a node sends a distance vector to its neighbors it does not send those routes which it learned from a neighbor back to that neighbor. A variation of the split horizon method is called *split horizon with poison reverse* where a node does send back routes learned from a neighbor back to that neighbor but advertises an infinite cost for such routes. Unfortunately, both the split horizon techniques work only for routing loops that involve two nodes.

Link State Routing

In link state routing, every node constructs a packet containing the cost of reaching its immediate neighbors and sends it to all the nodes in the network. Once again the assumption is that each node can discover its immediate neighbors and the costs to each of them. The packet which is sent by each node is called its *link state packet (LSP).* Once a node has received the LSPs from all the other nodes in the network it will have enough knowledge to build a complete routing map of the network.

The difference between distance vector routing and link state routing is that in distance vector routing each node sends information only to its immediate neighbors but the information sent is contains the list of all nodes reachable from it while in link state routing each node sends information to all the other nodes in the network but the information sent is only about its immediate neighbors.

The method used to distribute the LSPs from each node to all the other nodes in the network is called *reliable flooding.* In reliable flooding, each node sends the LSP to all its immediate neighbors and the neighbors then forward the LSP to all their immediate neighbors and so on. Each node receiving an LSP makes sure that the LSP is not forwarded back on the link it was received on. For example, in the network shown in Figure 7.2a if node B receives an LSP from node A it will forward it only to nodes C and D. Nodes use acknowledgements and retransmissions to make sure that LSPs are received reliably.

Each LSP contains the following

- A unique identifier of the node which created the LSP
- A list of immediate neighbors of that node along with the cost of reaching each one of them
- A sequence number
- A time to live (TTL) for the LSP

The first two fields enable the calculation of the routes. The last two fields help make the flooding process reliable and efficient. Every time a node receives an LSP it decrements the

TTL field before forwarding it to its neighbors. This ensures that old link state information is eventually removed from the network. Each time a node generates a new LSP it increments the sequence number by one. This sequence number is used to ensure that all the nodes use the latest link state information to build their routing maps. For example, when node A receives node B's LSP it checks to see if it already has a copy of an LSP from node B. If it does not, it stores the LSP it just received. If it does have a copy, it compares the sequence numbers in the stored and received LSPs. If the received LSP has a lower sequence number it is dropped. If the the stored LSP has a lower sequence number it is replaced with the received LSP. As in distance vector routing, each node generates LSPs either when a periodic timer expires or when a change in topology occurs.

Once a node has all the LSPs from all the other nodes it will be able to calculate the complete routing map of the network and use this map to decide the best route to each destination. The calculation of the routes is done using *Dijkstra's shortest-path algorithm.* In graph-theoretic terminology, the algorithm can be explained in the following manner. Let N be the set of nodes in the graph and let $l(A, B)$ be the cost associated with the edge between any nodes A and B in the network. We set $l(A, B) = \infty$ if no edge connects nodes A and B. Let $S \in N$ be a node which has received the LSPs from all the nodes in the network. We want to calculate $C_S(X)$, the cost of the shortest path from S to X for all nodes $X \in N$. The algorithm is as follows:

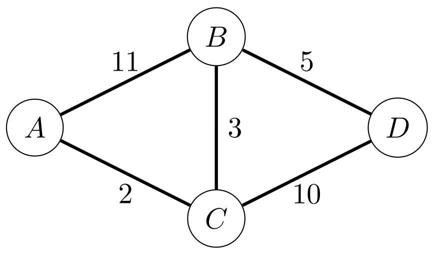
```
$M = \{S\}$
for each $X$ in $N - \{S\}$
    $C_S(X) = l(S, X)$
    if $C_S(X) < \infty$, next hop for $X$ is $X$ itself
while $(N \neq M)$
    $M = M \cup \{Y\}$ such that $C_S(Y)$ is the minimum among all $Y$ in $(N - M)$
    for each $X$ in $(N - M)$
        $C_S(X) = \min\{C_S(X), C_S(Y) + l(Y, X)\}$
        if $C_S(X)$ has changed, next hop for $X$ is the next hop to reach $Y$ from $S$
```

At any point in the execution of the above algorithm, M denotes the set of nodes which have been incorporated in the calculation of the shortest paths from S to all the other nodes in the network. The algorithm begins by setting M to be equal to S and then initializing the costs of the shortest paths using the known costs to the immediate neighbors. During this initialization, $C_S(X) = \infty$ for any X which is not an immediate neighbor of S. Then the node Y which is reachable with the lowest cost from S and is not yet in M is added to M. The costs of reaching all the nodes in $N - M$ are updated using the cost of reaching nodes through Y if this cost is smaller than the previous cost calculated. We also need to keep track of the next hop nodes as the algorithm progresses.

For example, consider the four node network shown in Figure 7.3a. Suppose node A has received LSPs from the the nodes B, C and D. Figure 7.3b shows the evolution of the link state routing table as node A runs Dijkstra's algorithm. In the table, the last three columns contain the cost and next hop for reaching the nodes B, C and D respectively. The second and third columns contain the sets M and $N - M$ at each step in the algorithm.

Interdomain Routing

An internetwork which is under the control of a single administrative entity is called an *autonomous system (AS).* An AS is also called a *routing domain* because the administrative entity



(a) A four-node communication network

Step	M	$N - M$	B	C	D
1	$\{A\}$	$\{B, C, D\}$	11,B	2,C	∞,-
2	$\{A, C\}$	$\{B, D\}$	5,C	2,C	12,C
3	$\{A, B, C\}$	$\{D\}$	5,C	2,C	10,C
4	$\{A, B, C, D\}$	$\{\}$	5,C	2,C	10,C

(b) Illustration of the evolution of the link state routing table of node A as it runs Dijkstra's algorithm

Figure 7.3: Illustration of link state routing

is free to choose the routing protocol which will run within the AS. The distance vector and link state routing protocols are *intradomain* routing protocols because they are used for building the routing tables within a routing domain. Another name for intradomain routing protocols is *interior gateway protocols (IGPs)*.

The Internet is organized as autonomous systems and routing between them is accomplished by *interdomain* routing protocols. While intradomain routing was focussed on finding the shortest paths between nodes, interdomain routing is concerned with finding a *loop-free policy-compliant* path to destinations. A policy is a set of rules which an AS abides by regarding the traffic it will send or receive. For example, a policy for an AS X which is connected to AS Y and AS Z can be the following. It prefers to send traffic through AS Y over AS Z but it will use AS Z if it is the only option. It will never carry traffic from AS Y to AS Z or vice versa. This kind of a policy would be typical of a corporation which has subscribed to Internet service from two Internet service providers (ISPs), one which is the primary ISP and the other is the backup. Since the corporation is a paying customer it will not help out the ISPs by transferring traffic between them using its own network. Since each intradomain routing protocol has its own metric for calculating optimal paths, it is not feasible for an interdomain routing protocol to aim for optimality in paths which span routing domains. Consequently, interdomain routing aims for reachability rather than optimality.

There have been two interdomain routing protocols which have been developed for the Internet. The first one was called the *Exterior Gateway Protocol (EGP)* and required the interconnections among the ASs to have a tree-like structure. This was a severe limitation of EGP and it was replaced by the *Border Gateway Protocol (BGP)* which is used in the Internet today. BGP works by having each AS configure one of its routers to be the *BGP speaker*. The BGP speakers of different ASs establish BGP sessions to exchange reachability information between the ASs. BGP advertises complete paths in terms of ASs to reach networks rather than a next hop. This helps check the policy compliance of a candidate path and also in detecting loops. A 16 bit identifier is assigned to an AS by a central authority to identify it.

7.3 Congestion Avoidance

A network is said to be congested if buffer overflows occur frequently in its nodes. While flow control is focussed on preventing a single source from overflowing the buffer at a destination, congestion control prevents a set of sources from causing buffer overflows anywhere in the network. Congestion control algorithms attempt to reduce and recover from congestion once it occurs while congestion avoidance algorithms try to predict the occurrence of congestion and take steps to prevent it from occurring. Congestion avoidance schemes require steps to be taken at both the network and transport layers but in this section we describe the network layer functionality of some such schemes.

Random Early Detection (RED)

In this scheme, each router in the network monitors the length of its queue and randomly drops a packet when the queue length is close to causing a buffer overflow. The reasoning behind this scheme is that the source which sent the packet will timeout waiting for an acknowledgement and infer that the network is congested or nearing congestion. It will then reduce its transmission rate. If a sufficient number of source reduce their transmission rates, congestion will be avoided. Of course, RED requires cooperation form the transport layer which will need to reduce the source transmission rate upon timeout.

The calculation of the queue length in RED involves a weighted running average, i.e. it is calculated as

$$L_{avg}(n+1) = (1-w) \times L_{avg}(n) + w \times L_{inst}(n+1)$$

where $L_{inst}(n)$ is the nth sample of the instantaneous value of the queue length, $L_{avg}(n)$ is the average value of the queue length after the nth sample is taken, and w is the weighting factor where $0 < w < 1$. The reason for using the average value of the queue length instead of the instantaneous value to predict imminent congestion is that bursts of packets which arrive at the router and serviced by it without causing overflows will not trigger the congestion avoidance mechanism. Such bursts will increase the instantaneous queue length but will not significantly affect the average queue length.

The probability of dropping a packet varies with the queue length in the following manner. RED maintains two thresholds for the average queue length: K_{min} and K_{max}. When a packet arrives at a router which is running RED, the average queue length L_{avg} is calculated and compared to these thresholds. If $L_{avg} \leq K_{min}$, the packet is queued. If $L_{avg} \geq K_{max}$, the packet is dropped. If $K_{min} < L_{avg} < K_{max}$, the packet is dropped with some probability P_{drop}. The exact value of P_{drop} as a function of these parameters is

$$\begin{aligned} P_{temp} &= P_{max} \times \frac{L_{avg} - K_{min}}{K_{max} - K_{min}} \\ P_{drop} &= \frac{P_{temp}}{1 - m \times P_{temp}} \end{aligned}$$

where m is the number of newly arrived packets which have been queued and not dropped while L_{avg} has been between the two thresholds. The motivation behind the second equation above is that as m increases P_{drop} increases making a packet drop increasingly likely as the time since the last drop increases. This made the packet drop events to be widely spaced in time. For example, consider a value of 0.02 for P_{max} and let $m = 0$ initially. If the average queue length $L_{avg} = \frac{K_{max}+K_{min}}{2}$, then the initial value of $P_{temp} = 0.01$. Since $m = 0$, P_{drop} is also equal to 0.01. A packet which arrives at the router has a 1% chance of being dropped and with arriving packet which is not dropped m increases causing P_{drop} to increase. Once 50 packets arrive without being dropped, P_{drop} increases to 0.02 and once 99 packets arrive without being dropped P_{drop} increases to 1. The nice feature of this algorithm is that it ensures that the packet drops are evenly distributed in time.

Explicit Congestion Notification (ECN)

In the previous scheme, the source was implicitly informed of impending congestion by dropping one of the packets sent by it. ECN is used to explicitly inform the source of impending congestion without dropping packets. In ECN, a router sets a bit in the IP header of an arriving packet when it is about to experience congestion. This bit is echoed back to the source by the destination of the packet. The source then reduces its transmission rate alleviating the congestion at the router. Two bits in the TOS field of the IPv4 header are used to implement ECN. One is set by the source to indicate that it is ECN-capable, i.e. it is capable of reacting to a congestion notification. The other bit is set by the routers along the path to the destination to indicate that congestion is about to occur.

Chapter 8

Transport Layer

The goal of the transport layer is to convert the host-to-host packet delivery service provided by the network layer into a process-to-process communication channel, i.e. it enables application processes running on the source node to communicate with application processes running on the destination node. The network layer, as epitomized by IP, provides a best-effort packet delivery service which is unreliable and has the following limitations:

- Packets can be dropped
- Packets can arrive out of order at the destination
- Duplicate copies of a packet can be delivered at a destination
- Packet size is limited by the minimum MTU along the path to the destination
- Packets may be delivered after a long delay

The transport layer has to implement algorithms to convert this underlying best-effort service into a reliable packet delivery service which has the following desirable characteristics:

- Guaranteed packet delivery
- Delivery of packets in the same order they were sent
- Delivery of at most one copy of a packet to the destination
- Support the delivery of arbitrarily large messages from source to destination
- Allow the destination to apply flow control on the source
- Allow multiple application processes on both source and destination hosts to communicate using the same network path

Different transport layer protocols provide different combinations of these desirable characteristics. We will discuss two transport layer protocols from the TCP/IP suite of protocols which are widely used on the Internet: *User Datagram Protocol (UDP)* and *Transmission Control Protocol (TCP)*.

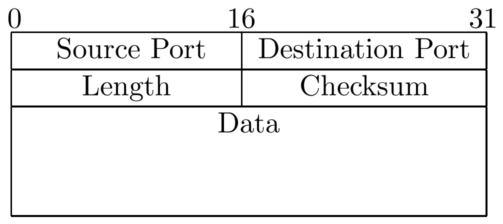


Figure 8.1: UDP header format

8.1 User Datagram Protocol (UDP)

UDP is the simplest transport layer protocol which builds on the end-to-end packet delivery service provided by IP into a process-to-process communication service. It achieves this by providing a way to demultiplex the packets arriving on a single network path to different application processes on the destination node.

The main problem which UDP solves is that there may be multiple application processes running on a destination which are receiving packets on the same link. All the packets will have the same destination IP address so there needs to be a mechanism to differentiate packets destined for different application processes. One way is to use the process id (pid) assigned by the operating system to the application process to identify it but this would require the source to learn the pid which is not feasible if the source is contacting the destination for the first time. A more commonly used approach is for the application processes on the source and destination to identify each other using an abstract identifer called the *port* which is defined beforehand for well-known applications. New applications for which a port has not been preassigned can use a well-known port to agree on a unused port for future communication. A port in this context is just a number and has no relation to the physical ports on the source and destination nodes. So an application running on a node is identified by an IP address and port where the IP address is the node's IP address. This (IP address, port) pair is called a *socket*.

The UDP header format is shown in Figure 8.1. The first two fields in the header correspond to the source and destination port identifier. Each identifier is 16 bits long so there can upto 65,536 ports. However, a port has to be uniquely assigned to an application process only on a single host since the application is identified by an (IP address, port) pair. The next header field gives the length of the UDP packet including the length of the header. Although UDP does not guarantee reliable delivery it does perform an elementary check on the integrity of the message using a 16-bit Internet checksum. The UDP checksum is optional in IPv4 but it will become mandatory in IPv6 because the IPv6 header does not have a checksum of its own. UDP computes the checksum over the UDP header, the UDP data and a set of fields from the IP and UDP headers which are called the *pseudoheader*. The pseudoheader consists of three fields from the IP header, namely the 8-bit protocol number, the source IP address and the destination IP address, and the UDP length field. The reason for including the pseudoheader in the checksum calculation is to ensure that the packet has been delivered to the correct destination and claims to be from the correct source. If the source or destination IP addresses are modified by intermediate routers which forward the packet, this is likely to be detected by the UDP checksum.

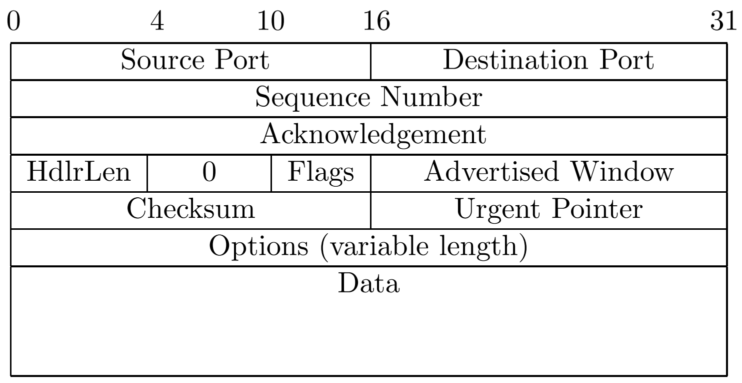


Figure 8.2: TCP segment structure

8.2 Transmission Control Protocol (TCP)

In contrast to UDP, TCP is a more complex transport layer protocol which offers a reliable, connection-oriented service between application processes running on the host and destination nodes. TCP has the following features

- It guarantees reliable, in-order delivery of a stream of bytes
- It is a full-duplex protocol in the sense that a single TCP connection supports a pair of byte streams, one flowing in each direction between source and destination nodes
- It includes a flow control mechanism which allows the receiver to control how much data a sender can transmit at a given time
- It supports a demultiplexing mechanism like UDP which allows multiple application processes on a host to receive packets on the same network path
- It also implements a congestion control algorithm which enables the source to react to network congestion and help alleviate it

TCP Segment Format

The packet exchanged between nodes using TCP are called *segments*. The structure of a TCP segment is shown in Figure 8.2. The segment consists of the following fields.

- **Source Port** The 16-bit source port number identifies the application on the source host which originated the TCP segment. The destination host can use this port number to send a reply to the source application which sent the segment.
- **Destination Port** The 16-bit destination port number identifies the application on the destination host the segment is destined for.
- **Sequence Number** The 32-bit sequence number field contains the sequence number of the first byte of data contained in the TCP segment. For instance, a segment with sequence number 101 and containing 200 bytes of data will be followed by a segment which has sequence number 301 in its sequence number field.

- **Acknowledgement** The 32-bit acknowledgement field contains the sequence number of the next expected byte. For instance, if the acknowledgement field contains the value 2001 then this means that the bytes having sequence number upto and including 2000 have been received correctly by the host sending the acknowledgement.
- **Header Length** The 4-bit header length field is used to indicate the length of the TCP header in multiples of 32-bit words.
- **Reserved** The 6 bits following the header length field are reserved for future use.
- **Flags** The 6-bit flag field is used to communicate control information to the destination. The flag bits in order of appearance are
  - **URG** Urgent pointer field valid
  - **ACK** Acknowledgement field valid
  - **PSH** Push operation invoked
  - **RST** Connection to be reset
  - **SYN** Start of a new connection
  - **FIN** Final segment from sender

  The SYN and FIN flags are used when opening and closing a connection, respectively. The ACK flag is set when the acknowledgement field is valid and indicates to the receiver that it should read the acknowledgement field. The URG flag is set when the segment contains urgent data. A segment with the PSH flag set forces the TCP process at the destination flush its buffers and pass the information to the application. If segments arrive without the PSH flag set, the destination can coalesce multiple segments before passing them to the application. The RST flag is used by the sending source to immediately reset or abort a connection because it has an unexpected error condition. Multiple flags can be set at the same time. For example, a segment with both the ACK and FIN flags set indicates that it is the last segment from the source host which is also acknowledging some received segments.
- **Advertised Window** The 16-bit advertised window field is used for flow control purposes. It quantifies the available buffer space at a receiver.
- **Checksum** The 16-bit checksum field is computed as the Internet checksum over the TCP header, the TCP payload, and a pseudoheader which contains the source and destination IP addresses as well as the length field of the IP header.
- **Urgent Pointer** The 16-bit urgent pointer points to the *end* of the urgent data. The urgent data is usually present immediately after the header so that it can be processed quickly. The URG flag is set for a segment which contains urgent data.
- **Options** TCP supports several options which are used to provide extra functionality. Each option is preceded by two bytes - the first byte contains the option type and the second byte indicates the length of the option in bytes (including the first two bytes). An example of an option is the maximum segment size (MSS) option which is used by the source host to negotiate the maximum size of the segment which will be used by the connection. The MSS is stored in a 16 bit field limiting it to 64 KB.
- **Data** The payload of the TCP segment is located at an offset specified by the header length field from the beginning of the TCP segment.

TCP Connection Establishment and Termination

The TCP connection establishment procedure between two hosts involves a *three-way handshake.* The host which initiates the connection is called the *client* and the host which responds to the client's request is called the *server.* The procedure is as follows.

1. The client sends a segment to the server with the SYN flag set and an initial segment number it plans to use for this TCP connection. Let the initial sequence number the client plans to use be x.
2. The server responds with a segment that has both the SYN and ACK flags set and includes a sequence number of its own which it plans to use for this connection. The acknowledgement field contains $x + 1$ which is the sequence number of the next byte which the server expects from the client. Let the initial sequence number the server plans to use be y.
3. Finally, the client sends a segment with the ACK flag set and containing the value $y + 1$ in the acknowledgement field.

Once these three segments have been exchanged, the TCP connection is established and the each host knows the initial sequence number the other host plans to use for this connection. These initial sequence numbers are randomly chosen rather than starting from a predetermined value like zero. The reason for doing this is to prevent delayed segments from a previous connection between the client and server from interfering with the current connection.

The TCP connection termination procedure involves a *four-way handshake.* This allows the client and the server to independently stop sending segments while allowing the other host to keep sending segments. The procedure is as follows.

1. The client sends a segment with the FIN flag set to indicate to the server that it wants to terminate the TCP connection.
2. The server sends a segment with the ACK flag set to confirm the receipt of the FIN segment. Once this acknowledgement is received by the client, it stops sending data segments in the client-to-server direction. However, the server may still need to send segments to the client and the client will need to respond to the received segments with acknowledgements.
3. When the server is ready to close the connection, it sends a segment with the FIN flag set.
4. When the client receives the FIN segment it responds with an ACK segment and the connection is terminated in both directions.

In the above description of the connection termination procedure, the role of the client and server can be interchanged, i.e. the server may send the FIN segment first.

TCP Acknowledgements

TCP guarantees reliable delivery of segments using retransmissions and acknowledgements. The TCP acknowledgement mechanism has the following features.

- **Cumulative Acknowledgements** When the acknowledgement field in the TCP segment contains a number $x+1$ it is acknowledging the correct receipt of all bytes having sequence number upto and including x. The advantage of this scheme is that lost acknowledgements do not cause a problem as long as subsequent acknowledgements with a higher sequence number are received correctly.
- **Piggybacking** When a receiver wants to acknowledge a correctly received TCP segment it can either send an ACK-only segment which consists of only the TCP header and has no data in it, or it can send the acknowledgement in a data segment travelling in the reverse direction. This method is called *piggybacking*. The advantage of piggybacking is that it reduces acknowledgement traffic in the reverse direction.
- **Delayed ACK** The TCP process on the receiving side has the option of generating an ACK as soon as a segment arrives or delaying the ACK for a while. If the ACK generation is delayed, then the receiver can possibly acknowledge the receipt of more than received segments with a single ACK and reduce traffic in the reverse direction. However, delaying an ACK too much can cause a timeout at the sending side and result in unnecessary retransmissions.
- **Duplicate ACK** If a segment is lost and subsequent segments arrive at the receiver, the receiving TCP process generates duplicate ACKs acknowledging the receipt of the bytes received so without gaps. For instance, suppose all the bytes upto sequence number x have been received correctly at the receiver and a segment containing bytes $x+1$ to y is lost. If segments with bytes having sequence number $y+1$ or later are received correctly at the receiver, duplicate acknowledgements will be generated which will have the number $x+1$ in the acknowledgement field. This mechanism is used by the sender to infer the loss of a segment.

TCP Sliding Window

TCP uses a sliding window protocol to implement a flow control mechanism between the sender and receiver. The receiver advertises the size of the sliding window to the sender using the 16-bit advertised window field in the TCP header. The maximum number of unacknowledged bytes the sender can have is limited by the value of the advertised window. The receiver chooses the value of the advertised window depending on the amount of available buffer space. This prevents the sender by causing buffer overflow at the receiver.

The sender maintains a send buffer which contains the following.

- Data which has been sent but has not been acknowledged
- Data which the sending application has written to the buffer but has not been transmitted yet

Three pointers are required at the sender side to indicate the boundaries of the different types of data - *LastByteAcked*, *LastByteSent*, and *LastByteWritten*. These are illustrated in Figure 8.3a. The figure shows the sequence of bytes processed by the sender side TCP as a horizontal tape. The shaded region of the tape is the bytes contained in the send buffer. Since a byte in the send buffer can be acknowledged only after it has been sent, we have the following inequality.

$$\text{LastByteAcked} \leq \text{LastByteSent}$$

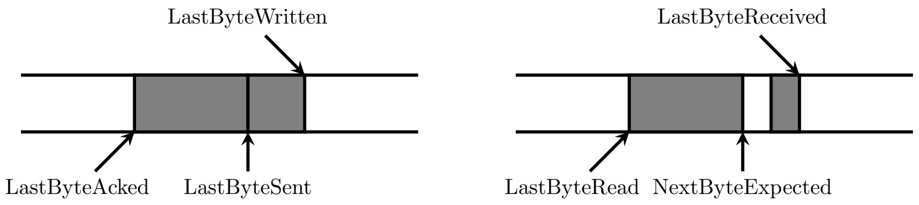


Figure 8.3: Illustration of the (a) TCP send buffer and (b) TCP receive buffer

Similarly a byte cannot be sent before the sending application has written it into the send buffer, we have the following inequality.

$$\text{LastByteSent} \leq \text{LastByteWritten}$$

Once a byte has been acknowledged it is no longer necessary to store it in the send buffer, i.e. all bytes to the left of and including LastByteAcked need not be stored in the send buffer. Similarly there are no bytes to the right of LastByteWritten in the send buffer because they have not been written into the buffer by the sending application. The maximum size of the send buffer is given by the parameter *MaxSendBuffer*.

The receiver maintains a receive buffer which contains the following.

- Data which has been received out of order
- Data which has been received in order but has not been read by the receiving application

Three pointers are required at the receiver side to indicate the boundaries of the different types of data - *LastByteRead*, *NextByteExpected*, and *LastByteReceived*. These are illustrated in Figure 8.3b. The gap in the shaded region illustrates a situation where bytes have arrived out of order. Since a byte cannot be read by the receiving application until it is received and all the bytes preceding it have also been received, we have the following inequality.

$$\text{LastByteRead} < \text{NextByteExpected}$$

If the data bytes arrive in order, NextByteExpected points to the byte after the byte pointed to by LastByteReceived. If the data bytes arrive out of order, NextByteExpected points to the start of the first gap in the data. Thus we have the following inequality.

$$\text{NextByteExpected} \leq \text{LastByteReceived} + 1$$

Once a byte has been read by the receiving application it is no longer necessary to store it in the receive buffer, i.e all bytes to the left of and including LastByteRead need not be stored in the receive buffer. There are no bytes to the right of LastByteReceived in the receive buffer because they have not been received.

The maximum size of the receive buffer is given by the parameter *MaxReceiveBuffer*. To avoid buffer overflow, the receiver side TCP must maintain the following inequality.

$$\text{LastByteReceived} - \text{LastByteRead} \leq \text{MaxReceiveBuffer}$$

The receiver advertises a window size of

$$\text{AdvertisedWindow} = \text{MaxReceiveBuffer} - (\text{LastByteReceived} - \text{LastByteRead})$$

which is an estimate of the amount of free space remaining in its buffer. As new data arrives at the receiver, the pointer LastByteReceived moves to the right reducing the size of the advertised window. As data is read by the receiving application, the pointer LastByteRead moves to the right reducing the size of the advertised window.

TCP on the sender side uses the advertised window sent by the receiver to maintain the following inequality.

$$\text{LastByteSent} - \text{LastByteAcked} \leq \text{AdvertisedWindow}$$

The maximum amount of new data the sender can transmit is given by the following equation.

$$\text{EffectiveWindow} = \text{AdvertisedWindow} - (\text{LastByteSent} - \text{LastByteAcked})$$

The sender side TCP must also ensure that the sending application does not overflow the send buffer by making sure the following inequality holds.

$$\text{LastByteWritten} - \text{LastByteAcked} \leq \text{MaxSendBuffer}$$

The sender side TCP may have to block the sending application from writing data whenever the above inequality is in danger of being violated.

It is now easy to see how a slow receiver prevents a fast sender from causing buffer overflow. If a receiver cannot read the received data quickly enough, the LastByteReceived pointer moves to the right faster than the LastByteRead pointer resulting in the AdvertisedWindow becoming zero. Then the sender cannot send any new data. Then the send buffer fills up and the sender side TCP blocks the sending application from writing any new data into the send buffer.

TCP Transmission Mechanism

The sending application writes bytes into the send buffer and it is up to the send side TCP process to decide when to send a TCP segment. If the flow control window is not restricting the sending of a segment, TCP has three mechanisms which trigger the transmission of a segment.

- TCP maintains a variable called the *maximum segment size (MSS)* which is the size of the largest segment which TCP can send without resulting in fragmentation by the IP on the sender. The MSS is typically equal to the MTU of the directly connected network minus the size of the TCP and IP headers. Whenever the sending application has written an MSS worth of bytes into the send side buffer, TCP sends a segment.
- TCP also sends a segment when the sending application explicitly asks it to do so by invoking a push operation.
- The third mechanism which triggers a segment transmission is *Nagle's algorithm* which will be described below.

Nagle's algorithm is a solution to a problem known as the *silly window syndrome*. Suppose that the flow control window on the sender side is currently closed. Now if an acknowledgement arrives from the receiver which increases the send window from zero to a value less than the MSS, say half the MSS. Then the send side TCP has two options: either send a segment of size equal to half the MSS or wait for an acknowledgement to arrive which will increase the window size to a full MSS. The former strategy of aggressively taking advantage of the available window size can result in the repeated transmission of very small or *silly* segments in which the data length is smaller than the header length. The problem is that once a small segment is introduced into the system it remains in the system until the connection terminates. Restricting TCP from ever sending small segments is not a feasible solution because an application can possibly invoke a push operation after it has written a single byte into the send buffer. Such a scenario is common in the case of interactive applications like Telnet. One solution is for the receiver to delay the acknowledgements corresponding to segments and acknowledge several of them at once at the same time. This delaying tactic can result in a larger window being advertised in the acknowledgement since it gives the receiving application more time to read data from the receive buffer. But a timeout can occur at the sender if the receiver delays the acknowledgements for too long. A better solution is to let the sender delay the sending of small segments according to Nagle's algorithm which is as follows.

```
When the sending application writes new data into the send buffer
    If the send window and data written are both greater than or equal to MSS
        Send a segment of size equal to MSS
    Else
        If there is unacknowledged data in flight
            Buffer the new data until an acknowledgement arrives
        Else
            Send all the new data immediately
```

Nagle's algorithm allows TCP to transmit a segment of size equal to the MSS if the data available to be sent and the effective send window are both at least as large as the MSS. But if either the data available or the effective send window are smaller than the MSS, TCP transmits a segment as long as there are no other unacknowledged segments in flight. If some of the previously sent segments have not been acknowledged, TCP waits for an acknowledgement to arrive before transmitting a segment. The consequence of Nagle's algorithm is that an application will send one segment per round-trip time (RTT). The arrival of the acknowledgement is used as the variable timer which triggers a tranmission. Using a clock-based timer to trigger transmissions would be harder to design for links having different and possibly time-varying RTTs. Nagle's algorithm alleviates but does not completely solve the silly window syndrome because the acknowledgement which triggers the transmission may not increase the effective send window to a large value.

TCP Timeout Computation

TCP uses timeouts and retransmissions to guarantee reliable delivery of segments. TCP sets the timeout value as a function of the RTT between a pair of hosts. However, estimating the RTT between two hosts on the Internet is not an easy problem given the range of possible RTTs between hosts and the possible variation in the RTT between the same two hosts over time. The RTT estimation algorithm used in TCP has evolved over the years as problems in previously proposed solutions became apparent.

The algorithm given in the original TCP specification used a running average of the RTT to estimate it and used the estimated value to compute the timeout. When TCP sends a segment it records the time. When it receives an acknowledgement for the segment it computes the difference between the current time and the segment transmission time as a sample RTT. It then estimates the RTT as a weighted average between the the previous estimate of the RTT and the sample RTT using the following equation.

$$\text{EstimatedRTT} = \alpha \times \text{EstimatedRTT} + (1 - \alpha) \times \text{SampleRTT}$$

The parameter α is chosen to be between 0 and 1. It controls the effect of temporary changes of the estimated RTT. A small value of α may result in the estimated RTT being heavily influenced by temporary fluctuations while a large value will suppress the effect of temporary fluctuations. The original TCP specification recommended a value between 0.8 and 0.9 for α. Once the RTT has been estimated, the timeout value is calculated as the following.

$$\text{TimeOut} = 2 \times \text{EstimatedRTT}$$

One problem with this method of estimating the RTT is that retransmissions can ruin the estimate. This is because when an acknowledgement arrives after a segment is retransmitted there is no way of knowing if the acknowlegment is for the first copy of the segment or the second copy of the segment. Suppose we associate the acknowledgement with the first copy of the segment. Then in the case that the first copy is corrupted the acknowledgement will actually be for the retransmitted segment and the difference between acknowledgement arrival time and the first copy's transmission time will be a wrong estimate of the sample RTT. If we associate the acknowledgement with the retransmitted segment, then an acknowledgement for the first copy may arrive after a long delay and cause an error in the estimation of the sample RTT. To deal with this problem, the *Karn/Partridge algorithm* was proposed in 1987. It has two main features, of which the first one specifically addresses the problem of RTT estimation in the presence of segment retransmissions.

- Whenever a segment retransmission happens due to a timeout, TCP does not estimate the RTT using the acknowledgements for such a segment. So TCP measures the sample RTT only for the segments which have been transmitted exactly once.
- Whenever a segment retransmission happens, TCP sets the next timeout value to twice the previous timeout value rather than basing it on the estimated RTT. This ensures that a TCP source does not retransmit too quickly in the presence of network congestion.

The RTT estimation algorithm was further improved in 1988 by taking the variance of the sample RTTs into account. The rationale behind this approach was that if the variation in the sample RTTs is small then the estimated RTT is close to the true RTT and timeout value can be chosen close to the estimated RTT rather than multiplying it by 2. But if the variation in the sample RTTs is large then the timeout value should be chosen conservatively. The following equations govern the calculation of the timeout by taking the variation into account.

$$\begin{aligned}
\text{Difference} &= \text{SampleRTT} - \text{EstimatedRTT} \\
\text{EstimatedRTT} &= \text{EstimatedRTT} + \delta \times \text{Difference} \\
\text{Deviation} &= \text{Deviation} + \delta(|\text{Difference}| - \text{Deviation}) \\
\text{TimeOut} &= \mu \times \text{EstimatedRTT} + \phi \times \text{Deviation}
\end{aligned}$$

where δ is a fraction between 0 and 1, μ is typically set to 1 and ϕ is typically set to 4. So when the variation in the sample RTTs is small, TimeOut is close to the EstimatedRTT while a large variation causes Deviation to dominate the TimeOut value.

TCP Congestion Control

In addition to the sliding window flow control algorithm, TCP implements a congestion control algorithm to prevent buffer overflows in the intermediate nodes along the path to the destination. TCP maintains a variable called the *CongestionWindow* which is used by the sender to limit the data being inserted into the network. This variable is the congestion control algorithm's counterpart of the flow control algorithm's advertised window. The TCP source now limits the maximum number of bytes of unacknowledged data to the minimum of the congestion window and the advertised window. TCP's effective window is now calculated in the following manner.

$$\begin{aligned}\text{MaxWindow} &= \text{MIN(CongestionWindow, AdvertisedWindow)}\\ \text{EffectiveWindow} &= \text{MaxWindow - (LastByteSent -LastByteAcked)}\end{aligned}$$

The main challenge in TCP congestion control as compared to TCP flow control is that nobody informs the source regarding the value of the CongestionWindow while the value of the AdvertisedWindow is sent to the source by the destination. The source has no option but to estimate the value of CongestionWindow by observing network events like timeouts in response to its previous transmissions. TCP interprets timeouts to be the result of network congestion and decreases the value of CongestionWindow when it encounters them. When timeouts do not occur, TCP increases the value of CongestionWindow. The exact mechanism used by TCP to modify CongestionWindow is called *additive increase/multiplicative decrease (AIMD).*

In AIMD, when a timeout occurs, the source sets CongestionWindow to half of its previous value. This halving corresponds to the "multiplicative decrease" part of AIMD. Although, the CongestionWindow variable represents a certain number of bytes it is easier to think of it in terms of packets for the purpose of illustration. Suppose the current value of CongestionWindow is 16 packets. If a timeout occurs, CongestionWindow is set to 8 packets. More timeouts will cause the value of CongestionWindow to be set to 4, then 2 and finally to 1 packet. The value of CongestionWindow is not allowed to go below 1 packet even if additional timeouts occurred. The size of the packet is typically the maximum segment size (MSS).

TCP's congestion control algorithm increases the value of CongestionWindow by 1 packet whenever the source receives acknowledgements for CongestionWindow number of packets. So if the current value of CongestionWindow is 4 and the source receives 4 acknowledgements, CongestionWindow is increased to 5. This is the "additive increase" part of AIMD.

AIMD is the right approach if the source is operating close to the capacity of the network path from itself to the destination. However, the additive increase method takes a long time to reach the capacity of the network. TCP provides a second mechanism called *slow start* to approach the full capacity of the link much faster by increasing the congestion window exponentially. The slow start algorithm is as follows.

1. Source starts by setting CongestionWindow to 1 packet
2. CongestionWindow is increased by 1 packet every time an acknowledgement arrives
3. If a timeout occurs, the following steps occur.
   (a) The current value of the CongestionWindow is halved and stored in a variable called the CongestionThreshold.
   (b) CongestionWindow is set to 1 packet and source enters slow start again until the value of CongestionWindow reaches CongestionThreshold.

(c) CongestionWindow then increases linearly according to AIMD. This phase is called *congestion avoidance*. A timeout in the congestion avoidance phase results in the CongestionWindow being halved and then increasing linearly for every reception of a CongestionWindow's worth of packets.

So if the current value of CongestionWindow is 4 and an acknowledgement arrives, CongestionWindow is increased to 5. Another acknowledgement arrival increases the value of CongestionWindow to 6. So by the time 4 acknowledgements arrive, the value of CongestionWindow is increased to 8. So the CongestionWindow has doubled after 4 acknowledgements while it increased to 5 in AIMD.

To understand what is "slow" about the slow start algorithm we need to compare it to the original behavior of TCP rather than the linear increase in AIMD. In the original implementation of TCP, when a connection was established and the source first started to send packets the advertised window was at a very large value because the receiver's buffer was empty. If the source inserted the advertised window's worth of bytes into the network, the intermediate routers along the path to the destination would not be able to handle this burst of traffic. So slow start was designed to space the sending of packets to prevent a sudden burst of traffic from overflowing intermediate node buffers at the same time reaching the capacity of the link quickly. Thus slow start is used in two situations (both of which have a large advertised window)

- At the very beginning of a TCP connection
- When a connection goes dead waiting for a timeout. This is usually because a packet was lost and the sender continued to send packets until its effective window dwindled to zero. The lost packet is eventually retransmitted resulting in a cumulative ACK which opens the effective window to a very large value. Now the source uses slow start to space the insertion of the large amount of data into the network.

Two more features were added to the TCP congestion control mechanism to improve the throughput in the presence of packet losses - *fast retransmit* and *fast recovery*. In the original TCP, a lost packet would be retransmitted only after a timeout. But the receiver would send an acknowledgement for every out-of-order packet repeatedly acknowledging the bytes which were successfully received just before the missing packet. Such acknowledgements are called *duplicate ACKs*. In fast retransmit, the source retransmits the missing packet as soon as it receives three duplicate ACKs. The identity of the missing packet is garnered from the next expected byte sequence number which is present in the duplicate ACKs. The reason for waiting for three duplicate ACKs before retransmitting the packet rather than doing so immediately after the first duplicate ACK is received is that duplicate ACKs can also arrive because of packet reordering by the network. Waiting for three duplicate ACKs is a tradeoff between avoiding unnecessary retransmissions and retransmitting a missing packet quickly. Since fast retransmit uses three duplicate ACKs to detect a missing packet, it also infers the occurrence of network congestion from this event. If this packet loss is detected during slow start, the congestion window would be set to 1 and slow start is run until the congestion threshold is reached. In fast recovery, this slow start phase between the loss of a packet and the eventual additive increase is eliminated and an additive increase strategy is immediately followed after halving the value of the congestion window.

In conclusion, slow start is used only at the beginning of a connection and when a timeout occurs. At all other times, the AIMD mechanism is used to modify the congestion window.