Q.1 a. Write the ways in which the OSI reference model and TCP/IP reference model are different and the ways they are the similar. (4)

Answer:
Both models are based on layered protocols. Both have a network, transport and application layer. In both models, the transport service can provide a reliable end-to-end byte stream. Differ in several ways: The number of layers is different, the TCP/IP does not have session or presentation layers. OSI does not support internetworking, and OSI has both connection-oriented and connectionless service in the network layer.

b. Consider the sliding window Go-Back-N ARQ system in which S sends packets 0,1,2,3,4,5 and 6. Packet 3 received at R is corrupted then what do S and R sends to each other next? (4)

Answer: R sends ACK-3 , S then sends packets 3,4,5,6,7,0 and 1.

c. Write name of OSI layer which is responsible for the following.
(i) To route the packets, determine the best path
(ii) Responsible to provide end-to-end communications with reliable service
(iii) Provides node-to-node reliable communications
(iv) Provides the congestion control (4)

Answer: i)The network layer ii) The transport iii The data link layer iv) Network layer

d. Write the key differences between IP addresses and MAC addresses. (4)

Answer:
MAC addresses are flat, whereas IP addresses are hierarchical , addresses are 48-bits long, whereas IPv4 addresses are 32-bits long. MAC addresses are globally unique, whereas IP addresses are not necessarily. MAC addresses are burned into the device, whereas IP addresses may be assigned dynamically. MAC addresses are used at the link layer within a single network, whereas IP addresses are used at the network layer between networks

e. What is the man-in-the-middle attack? Can this attack occur when symmetric keys are used? (4)

Answer:
In a man-in-the-middle attacker, the attacker interposes him/herself between the sender and receiver, often performing some transformation (e.g., re-encoding or altering) of data between the sender and receiver. Man-in-the-middle attacks can be particularly pernicious since the sender and receiver will each receive what the other has sent and since they are using encryption would think that they have achieved confidentiality.

f. Why was IPv6 created? Describe the most significant changes of IPv6 compared to IPv4. (4)

Answer:
IPv6 was primarily created to overcome the address shortage of IPv4 addresses. The most significant change is the increase in address length from 32 bits to 128 bits. Another change was the addition of a flow label.

g. E-mail clients in personal computers mostly use the POP or the IMAP protocols to download incoming mail. Why do they not use SMTP? (4)

Answer:

SMTP requires that the receiving program is on-line all the time, and personal computers are sometimes shut down or not running any mail program. POP and IMAP store incoming e-mail in the server, until the user downloads them. POP and IMAP identify the user before giving access to the e-mail for this user, which can give better security.

Q.2 a. A slotted ALOHA network transmits 200-bit frames using a shared channel with a 200-kbs bandwidth. Find the throughput if the system (all stations together) produces (5)
(i) 1000 frames per second (ii) 250 frames per second

Answer:

(i) Frame transmission time is 200/200 kbs=1ms

i.e system generate 1000 frame per second(one frame per second) So Load (G) is 1.

$S=G \times e^{-G}$ ==> S =0.368

Throughput is == 1000 x 0.368=368 frames i.e out of 1000 only 368 frames probably survive.

(ii) Here G=1/2 So $S=G \times e^{-G}$ ==> S =0.303

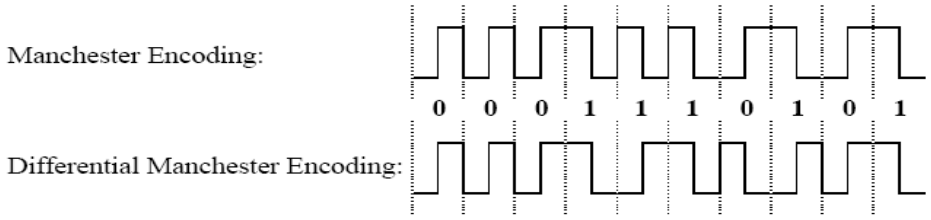
Throughput is == 500 x 0.303=151.5 frames i.e out of 500 only 151 frames probably survive.

b. What are fundamental differences which exist between NRZ-L and NRZI ? Sketch the Manchester encoding and Differential encoding for the bit stream: 0001110101? (6)

Answer:

Difference between NRZ-L and NRZI

– With NRZ-L, the receiver has to check the voltage *level* for each bit to determine whether the bit is a 0 or a 1,
– With NRZI, the receiver has to check whether there is a *change at the beginning* of the bit to determine if it is a 0 or a 1



c. Explain CSMA and its different versions. What are the limitations of each version? Briefly explain CSMA/CD and CSMA/CA. In which situation should we use each of the above? (7)

Answer: Page 250-254 (Tenenbaum)

With slotted ALOHA the best channel utilization that can be achieved is $1/e$. This is hardly surprising, since with stations transmitting at will, without paying attention to what the other stations are doing, there are bound to be many collisions. In local area networks, however it is possible for stations to detect what other stations are doing, and adapt their behavior accordingly. These networks can achieve a much better utilization than $1/e$. In this section we will discuss some protocols for improving performance.

Protocols in which stations listen for a carrier (i.e., a transmission) and act accordingly are called **carrier sense protocols**. A number of them have been proposed. Kleinrock and Tobagi (1975) have analyzed several such protocols in detail. Below we will mention several versions of the carrier sense protocols.

Persistent and Nonpersistent CSMA

The first carrier sense protocol that we will study here is called **1-persistent CSMA** (Carrier Sense Multiple Access). When a station has data to send, it first listens to the channel to see if anyone else is transmitting at that moment. If the channel is busy, the station waits until it becomes idle. When the station detects an idle channel, it transmits a frame. If a collision occurs, the station waits a random amount of time and starts all over again. The protocol is called 1-persistent because the station transmits with a probability of 1 whenever it finds the channel idle.

The propagation delay has an important effect on the performance of the protocol. There is a small chance that just after a station begins sending, another station will become ready to send and sense the channel. If the first station's signal has not yet reached the second one, the latter will sense an idle channel and will also begin sending, resulting in a collision. The longer the propagation delay, the more important this effect becomes, and the worse the performance of the protocol.

Even if the propagation delay is zero, there will still be collisions. If two stations become ready in the middle of a third station's transmission, both will wait politely until the transmission ends and then both will begin transmitting exactly simultaneously, resulting in a collision. If they were not so impatient, there would be fewer collisions. Even so, this protocol is far better than pure ALOHA, because both stations have the decency to desist from interfering with the third station's frame. Intuitively, this will lead to a higher performance than pure ALOHA. Exactly the same holds for slotted ALOHA.

A second carrier sense protocol is **nonpersistent CSMA**. In this protocol, a conscious attempt is made to be less greedy than in the previous one. Before sending, a station senses the channel. If no one else is sending, the station begins doing so itself. However, if the channel is already in use, the station does not continually sense it for the purpose of seizing it immediately upon detecting the end of the previous transmission. Instead, it waits a random period of time and then repeats the algorithm. Intuitively this algorithm should lead to better channel utilization and longer delays than 1-persistent CSMA.

The last protocol is **p-persistent CSMA**. It applies to slotted channels and works as follows. When a station becomes ready to send, it senses the channel. If it is idle, it transmits with a probability p. With a probability $q = 1 - p$ it defers until the next slot. If that slot is also idle, it either transmits or defers again, with probabilities p and q. This process is repeated until either the frame has been transmitted or another station has begun transmitting. In the latter case, it acts as if there had been a collision (i.e., it waits a random time and starts again). If the station initially senses the channel busy, it waits until the next slot and applies the above algorithm. Figure 4-4 shows the throughput versus offered traffic for all three protocols, as well as pure and slotted ALOHA.

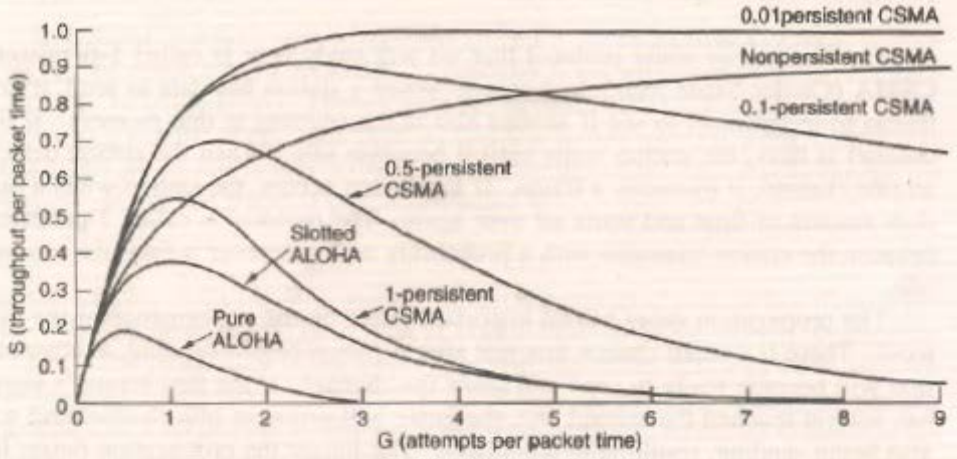


Fig. 4-4. Comparison of the channel utilization versus load for various random access protocols.

CSMA with Collision Detection

Persistent and nonpersistent CSMA protocols are clearly an improvement over ALOHA because they ensure that no station begins to transmit when it senses the channel busy. Another improvement is for stations to abort their transmissions as soon as they detect a collision. In other words, if two stations sense the channel to be idle and begin transmitting simultaneously, they will both detect the collision almost immediately. Rather than finish transmitting their frames, which are irretrievably garbled anyway, they should abruptly stop transmitting as soon as the collision is detected. Quickly terminating damaged frames saves time and bandwidth. This protocol, known as **CSMA/CD** (**Carrier Sense Multiple Access with Collision Detection**), is widely used on LANs in the MAC sublayer.

CSMA/CD, as well as many other LAN protocols, uses the conceptual model of Fig. 4-5. At the point marked t_0, a station has finished transmitting its frame. Any other station having a frame to send may now attempt to do so. If two or more stations decide to transmit simultaneously, there will be a collision. Collisions can be detected by looking at the power or pulse width of the received signal and comparing it to the transmitted signal.

After a station detects a collision, it aborts its transmission, waits a random period of time, and then tries again, assuming that no other station has started transmitting in the meantime. Therefore, our model for CSMA/CD will consist of alternating contention and transmission periods, with idle periods occurring when all stations are quiet (e.g., for lack of work).

Now let us look closely at the details of the contention algorithm. Suppose

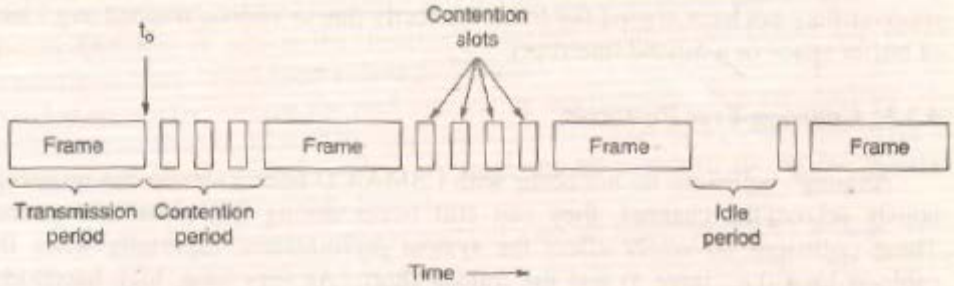


Fig. 4-5. CSMA/CD can be in one of three states: contention, transmission, or idle.

that two stations both begin transmitting at exactly time t_0. How long will it take them to realize that there has been a collision? The answer to this question is vital to determining the length of the contention period, and hence what the delay and throughput will be. The minimum time to detect the collision is then just the time it takes the signal to propagate from one station to the other.

Based on this reasoning, you might think that a station not hearing a collision for a time equal to the full cable propagation time after starting its transmission could be sure it had seized the cable. By "seized," we mean that all other stations knew it was transmitting and would not interfere. This conclusion is wrong. Consider the following worst-case scenario. Let the time for a signal to propagate between the two farthest stations be τ. At t_0, one station begins transmitting. At $\tau - \varepsilon$, an instant before the signal arrives at the most distant station, that station also begins transmitting. Of course, it detects the collision almost instantly and stops, but the little noise burst caused by the collision does not get back to the original station until time $2\tau - \varepsilon$. In other words, in the worst case a station cannot be sure that it has seized the channel until it has transmitted for 2τ without hearing a collision. For this reason we will model the contention interval as a slotted ALOHA system with slot width 2τ. On a 1-km long coaxial cable, $\tau \approx 5$ μsec. For simplicity we will assume that each slot contains just 1 bit. Once the channel has been seized, a station can transmit at any rate it wants to, of course, not just at 1 bit per 2τ sec.

It is important to realize that collision detection is an *analog* process. The station's hardware must listen to the cable while it is transmitting. If what it reads back is different from what it is putting out, it knows a collision is occurring. The implication is that the signal encoding must allow collisions to be detected (e.g., a collision of two 0-volt signals may well be impossible to detect). For this reason, special encoding is commonly used.

CSMA/CD is an important protocol. Later in this chapter we will study one version of it, IEEE 802.3 (Ethernet), which is an international standard.

To avoid any misunderstanding, it is worth nothing that no MAC-sublayer protocol guarantees reliable delivery. Even in the absence of collisions, the receiver may not have copied the frame correctly due to various reasons (e.g., lack of buffer space or a missed interrupt).

Q.3 a. Consider that a 48,000 bit packet is to be transmitted on a link having the propagation speed of 2 x 10^8 m/sec and physical link length of 1,000 m. Now suppose that the node can transmit at a rate of 4 Gbps. (6)
(i) What is the transmission time for the packet?
(ii) What is the propagation delay on the link?

(iii) Suppose that if the node starts transmitting the packet at time t=0, then at what time is the packet fully received at the destination?

Answer:

(i) 48,000 bits / 4 x 10^9 bps = 12 microsec

(ii) 1,000 m / 2 x 10^8 m/s = 5 microsec

(iii) It takes the transmission time to get the packet onto the link. Once there, it is not fully received until the last bit is received: this occurs after the propagation delay. The total time is therefore 12 microsec + 5 microsec = 17 microsec So at t=17 microsec the packet is fully received at the destination.

b. What is the key difference between a bridge and a repeater? Does a bridge achieve the same purpose similar to repeater? (6)

Answer:

A bridge is layer 2 and a repeater is layer 1. A repeater passes through all frames and regenerates the digital signal. When the signal is regenerated it is brought back to the "full" voltage level of the signal (e.g. as sent at the sending host). A bridge also regenerates the digital signal so it achieves the same purpose in this respect.

c. If a binary signal is sent over a 3-kHz channel whose signal-to-noise ratio is 20 dB, what is the maximum achievable data rate? (6)

Answer:

According to Shannon theorem which specifies the maximum data rate in a noisy channel as B*log_2^(1+S/N) where B is the bandwidth, S/N is the signal-to-noise ratio. Usually, S/N is given in "decibel", not just a ratio. **D**ecibel is calculated by dB=10log_10^(S/N) Therefore, we get S/N first as 20dB=10log_10^(S/N) ==> S/N=10^2=100

Substitute S/N into Shannon's theorem, we get the maximum bps as

3k*log2^(1+S/N)=3log2^101kbps=19.97kbps. However, the maximum data rate, assuming the channel is noise free, is only 6kbps, according to Nyquist rate. Therefore, the final answer should be 6kbps. Nyquist limit is 2 * 3 * log 2 b/s (base is 2)= 6kb/s

Shannon limit = 3 * log (1 + s/n) As signal - noise ratio is 20 db , we have s/n = 100

= 3 * log 101 = 19.85kbps

Thus, the limiting value is Nyquist limit i.e 6Kbps

Q.4 a. Consider a message D, presented by the following polynomial $x^{19} + x^{17} + x^{16} + x^{13} + x^{12} + x^{11} + x^9 + x^5 + x^2 + 1$, which is transmitted using the standard Cyclic Redundancy Check (CRC) method. The generator polynomial is $x^7 + x^5 + x^4 + x^3 + x^2 + 1$. Find the CRC and show the actual bit string to be transmitted. (7)

Answer: D = 10110011101000100101 G = 10111101 CRC bit: 0111 001

Transmitted bit: 10110011101000100101 0111 001

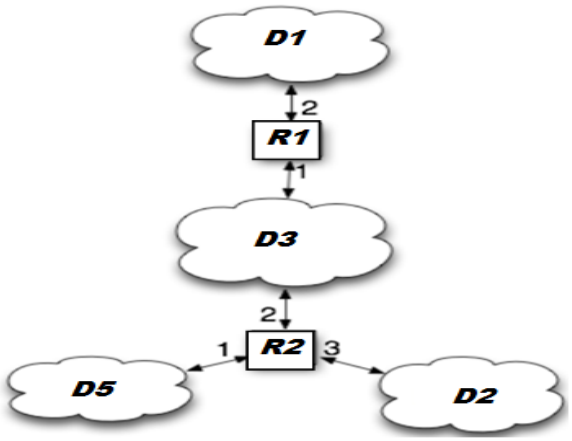
b. Write the hamming code of the data 1001101. Consider that receiver receives the code 10010100101. When it uses the hamming encoding method with even parity, then which bit(s) is in error? What is the correct code? (6)

Answer: hamming code: 10011100101 , 7th bit have error, correct bit: 1100 1110 111

c. Find out the 2-dimensional even bit parity check of the data 1100111 1011101 0111001 0101001. Write the data and parity bit which will be transmitted. (5)

Answer: data and parity bit: 11001111 10111011 01110010 01010011 **01010101**

Q.5 a. Suppose that you have assigned the task of designing a network in which you have to setup two routers R1 and R2 as shown below in the figure. You have assigned at your disposal the address 128.119.248.0/21. Consider that D1 Network has 1000 nodes, D2 Network has 200 nodes, D3 Network has 500 nodes and D5 network has 250 nodes; assign network addresses to each of the four sub network in the form a.b.c.d/x. Justify how this address space can be distributed in the domain by presenting the network identifiers for each of the four networks and also provide the forwarding tables for the two routers. (13)



Network	No. of Nodes
D1	1000
D2	200
D3	500
D5	250

Answer:

Network	No.of Nodes	Subnet	Here Bold bits showing host's bits
FD1	1000	128.119.252.0/22	*128.119.1111 1* 1 **0 0 .0000 0000**/22
FD2	200	128.119.248.0/24	*128.119.1111 1* 0 0 0 .**0000 0000**/24
FD3	500	128.119.250.0/23	*128.119.1111 1* 0 1 **0 .0000 0000**/23
FD5	250	128.119.249.0/24	*128.119.1111 1* 0 0 1 .**0000 0000**/24

Router 1's Forwarding Table

Destination Address	*Link Interface*
128.119.248.0/22	1
128.119.252.0/22	2

Router 2's Forwarding Table

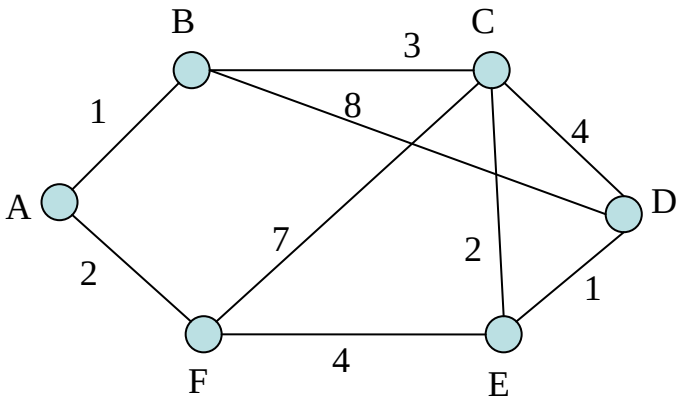
Destination Address	*Link Interface*
128.119.248.0/24	1
128.119.248.0/21 or 128.119.250.0/23 or 128.119.252.0/22	2
128.119.249.0/24	3

b. Consider an organization which is assigned the network address 193.1.1.0/24. Organization wants to create six subnets. The largest subnet is required to support 25 hosts. Write the subnet mask and subnet addresses of all the six subnets. (5)

Answer:

Network address : 193.1.1.0/24
193.1.1.0/27
193.1.1.32/27
193.1.1.64/27
193.1.1.96/27
193.1.1.128/27
193.1.1.160/27
193.1.1.192/27
193.1.1.224/27

Q.6 a. Consider the below given network topology. Show the iterations followed by node A in a tabular form to compute the shortest paths from itself to all the other nodes using the Dijkstra's algorithm. You can assume A has already collected all link state information in the network. (10)



Answer:

Step	S	D(B),p(	D(C),p(	D(D),p(	D(E),p(	D(F),p(
0	A	1,A	∞	∞	∞	2,A
1	AB		4,B	9,B	∞	2,A
2	ABF		4,B	9,B	6,F	
3	ABFC			8,C	6,F	
4	ABFCE			7,E		
5	ABFCE					

The forwarding table in A consists of columns for destination, nextHop and cost as follows:

destination	nextHop	Cost
A	--	--
B	B	1
C	B	4
D	F	7
E	F	6
F	F	2

b. With an example explain Link state routing and compare it with distance vector routing algorithm. (8)

Answer:

Modern computer networks generally use dynamic routing algorithms rather than the static ones described above. Two dynamic algorithms in particular, distance vector routing and link state routing, are the most popular. In this section we will look at the former algorithm. In the following one we will study the latter one.

Distance vector routing algorithms operate by having each router maintain a table (i.e, a vector) giving the best known distance to each destination and which line to use to get there. These tables are updated by exchanging information with the neighbors.

The distance vector routing algorithm is sometimes called by other names, including the distributed **Bellman-Ford** routing algorithm and the **Ford-Fulkerson** algorithm, after the researchers who developed it (Bellman, 1957; and Ford and Fulkerson, 1962). It was the original ARPANET routing algorithm and was also used in the Internet under the name RIP and in early versions of DECnet and Novell's IPX. AppleTalk and Cisco routers use improved distance vector protocols.

In distance vector routing, each router maintains a routing table indexed by, and containing one entry for, each router in the subnet. This entry contains two parts: the preferred outgoing line to use for that destination, and an estimate of the time or distance to that destination. The metric used might be number of hops, time delay in milliseconds, total number of packets queued along the path, or something similar.

The router is assumed to know the "distance" to each of its neighbors. If the metric is hops, the distance is just one hop. If the metric is queue length, the router simply examines each queue. If the metric is delay, the router can measure

it directly with special ECHO packets that the receiver just timestamps and sends back as fast as it can.

As an example, assume that delay is used as a metric and that the router knows the delay to each of its neighbors. Once every T msec each router sends to each neighbor a list of its estimated delays to each destination. It also receives a similar list from each neighbor. Imagine that one of these tables has just come in from neighbor X, with X_i being X's estimate of how long it takes to get to router i. If the router knows that the delay to X is m msec, it also knows that it can reach router i via X in $X_i + m$ msec via X. By performing this calculation for each neighbor, a router can find out which estimate seems the best and use that estimate and the corresponding line in its new routing table. Note that the old routing table is not used in the calculation.

This updating process is illustrated in Fig. 5-10. Part (a) shows a subnet. The first four columns of part (b) show the delay vectors received from the neighbors of router J. A claims to have a 12-msec delay to B, a 25-msec delay to C, a 40-msec delay to D, etc. Suppose that J has measured or estimated its delay to its neighbors, A, I, H, and K as 8, 10, 12, and 6 msec, respectively.

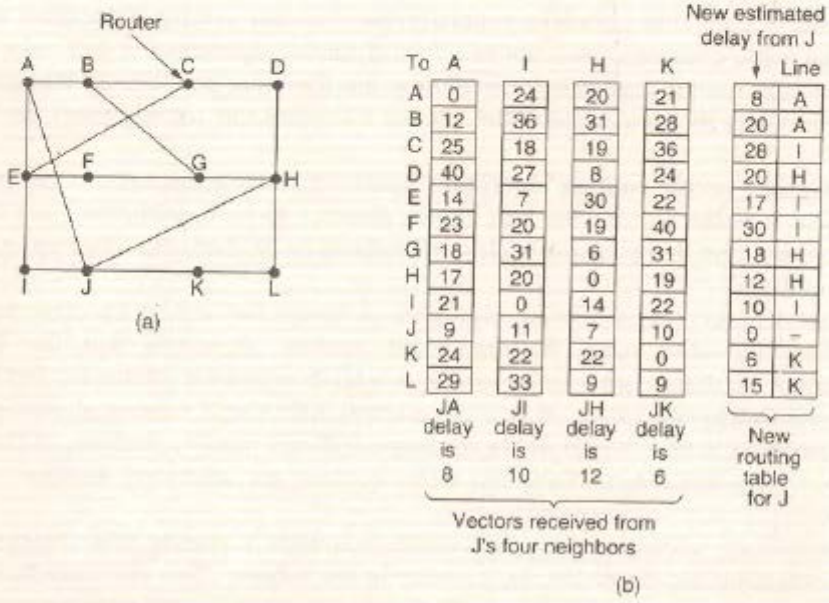


Fig. 5-10. (a) A subnet. (b) Input from A, I, H, K, and the new routing table for J.

Consider how J computes its new route to router G. It knows that it can get to A in 8 msec, and A claims to be able to get to G in 18 msec, so J knows it can count on a delay of 26 msec to G if it forwards packets bound for G to A.

Similarly, it computes the delay to G via I, H, and K as 41 (31 + 10), 18 (6 + 12), and 37 (31 + 6) msec respectively. The best of these values is 18, so it makes an entry in its routing table that the delay to G is 18 msec, and that the route to use is via H. The same calculation is performed for all the other destinations, with the new routing table shown in the last column of the figure.

The Count-to-Infinity Problem

Distance vector routing works in theory but has a serious drawback in practice: although it converges to the correct answer, it may do so slowly. In particular, it reacts rapidly to good news, but leisurely to bad news. Consider a router whose best route to destination X is large. If on the next exchange neighbor A suddenly reports a short delay to X, the router just switches over to using the line to A to send traffic to X. In one vector exchange, the good news is processed.

To see how fast good news propagates, consider the five-node (linear) subnet of Fig. 5-11, where the delay metric is the number of hops. Suppose A is down initially and all the other routers know this. In other words, they have all recorded the delay to A as infinity.

A	B	C	D	E	
	∞	∞	∞	∞	Initially
	1	∞	∞	∞	After 1 exchange
	1	2	∞	∞	After 2 exchanges
	1	2	3	∞	After 3 exchanges
	1	2	3	4	After 4 exchanges

(a)

A	B	C	D	E	
	1	2	3	4	Initially
	3	2	3	4	After 1 exchange
	3	4	3	4	After 2 exchanges
	5	4	5	4	After 3 exchanges
	5	6	5	6	After 4 exchanges
	7	6	7	6	After 5 exchanges
	7	8	7	8	After 6 exchanges
	⋮	⋮	⋮	⋮	
	∞	∞	∞	∞	

(b)

Fig. 5-11. The count-to-infinity problem.

When A comes up, the other routers learn about it via the vector exchanges. For simplicity we will assume that there is a gigantic gong somewhere that is struck periodically to initiate a vector exchange at all routers simultaneously. At the time of the first exchange, B learns that its left neighbor has zero delay to A. B now makes an entry in its routing table that A is one hop away to the left. All the other routers still think that A is down. At this point, the routing table entries for A are as shown in the second row of Fig. 5-11(a). On the next exchange, C learns that B has a path of length 1 to A, so it updates its routing table to indicate a path of length 2, but D and E do not hear the good news until later. Clearly, the good news is spreading at the rate of one hop per exchange. In a subnet whose longest

path is of length N hops, within N exchanges everyone will know about newly revived lines and routers.

Now let us consider the situation of Fig. 5-11(b), in which all the lines and routers are initially up. Routers B, C, D, and E have distances to A of 1, 2, 3, and 4, respectively. Suddenly A goes down, or alternatively, the line between A and B is cut, which is effectively the same thing from B's point of view.

At the first packet exchange, B does not hear anything from A. Fortunately, C says "Do not worry. I have a path to A of length 2." Little does B know that C's path runs through B itself. For all B knows, C might have ten outgoing lines all with independent paths to A of length 2. As a result, B now thinks it can reach A via C, with a path length of 3. D and E do not update their entries for A on the first exchange.

On the second exchange, C notices that each of its neighbors claims to have a path to A of length 3. It picks one of the them at random and makes its new distance to A 4, as shown in the third row of Fig. 5-11(b). Subsequent exchanges produce the history shown in the rest of Fig. 5-11(b).

From this figure, its should be clear why bad news travels slowly: no router ever has a value more than one higher than the minimum of all its neighbors. Gradually, all the routers work their way up to infinity, but the number of exchanges required depends on the numerical value used for infinity. For this reason, it is wise to set infinity to the longest path plus 1. If the metric is time delay, there is no well-defined upper bound, so a high value is needed to prevent a path with a long delay from being treated as down. Not entirely surprisingly, this problem is known as the **count-to-infinity** problem.

The Split Horizon Hack

Many ad hoc solutions to the count-to-infinity problem have been proposed in the literature, each one more complicated and less useful than the one before it. We will describe just one of them here and then tell why it, too, fails. The **split horizon** algorithm works the same way as distance vector routing, except that the distance to X is not reported on the line that packets for X are sent on (actually, it is reported as infinity). In the initial state of Fig. 5-11(b), for example, C tells D the truth about the distance to A, but C tells B that its distance to A is infinite. Similarly, D tells the truth to E but lies to C.

Now let us see what happens when A goes down. On the first exchange, B discovers that the direct line is gone, and C is reporting an infinite distance to A as well. Since neither of its neighbors can get to A, B sets its distance to infinity as well. On the next exchange, C hears that A is unreachable from both of its neighbors, so it marks A as unreachable too. Using split horizon, the bad news propagates one hop per exchange. This rate is much better than without split horizon.

The real bad news is that split horizon, although widely used, sometimes fails.

Consider, for example, the four-node subnet of Fig. 5-12. Initially, both *A* and *B* have a distance 2 to *D*, and *C* has a distance 1 there.

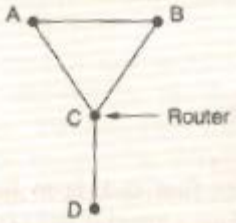


Fig. 5-12. An example where split horizon fails.

Now suppose that the *CD* line goes down. Using split horizon, both *A* and *B* tell *C* that they cannot get to *D*. Thus *C* immediately concludes that *D* is unreachable and reports this to both *A* and *B*. Unfortunately, *A* hears that *B* has a path of length 2 to *D*, so it assumes it can get to *D* via *B* in 3 hops. Similarly, *B* concludes it can get to *D* via *A* in 3 hops. On the next exchange, they each set their distance to *D* to 4. Both of them gradually count to infinity, precisely the behavior we were trying to avoid.

5.2.6. Link State Routing

Distance vector routing was used in the ARPANET until 1979, when it was replaced by link state routing. Two primary problems caused its demise. First, since the delay metric was queue length, it did not take line bandwidth into account when choosing routes. Initially, all the lines were 56 kbps, so line bandwidth was not an issue, but after some lines had been upgraded to 230 kbps and others to 1.544 Mbps, not taking bandwidth into account was a major problem. Of course, it would have been possible to change the delay metric to factor in line bandwidth, but a second problem also existed, namely, the algorithm often took too long to converge, even with tricks like split horizon. For these reasons, it was replaced by an entirely new algorithm now called **link state routing**. Variants of link state routing are now widely used.

The idea behind link state routing is simple and can be stated as five parts. Each router must

1. Discover its neighbors and learn their network addresses.
2. Measure the delay or cost to each of its neighbors.
3. Construct a packet telling all it has just learned.
4. Send this packet to all other routers.
5. Compute the shortest path to every other router.

In effect, the complete topology and all delays are experimentally measured and distributed to every router. Then Dijkstra's algorithm can be used to find the shortest path to every other router. Below we will consider each of these five steps in more detail.

Learning about the Neighbors

When a router is booted, its first task is to learn who its neighbors are. It accomplishes this goal by sending a special HELLO packet on each point-to-point line. The router on the other end is expected to send back a reply telling who it is. These names must be globally unique because when a distant router later hears that three routers are all connected to *F*, it is essential that it can determine whether or not all three mean the same *F*.

When two or more routers are connected by a LAN, the situation is slightly more complicated. Fig. 5-13(a) illustrates a LAN to which three routers, *A*, *C*, and *F*, are directly connected. Each of these routers is connected to one or more additional routers, as shown.

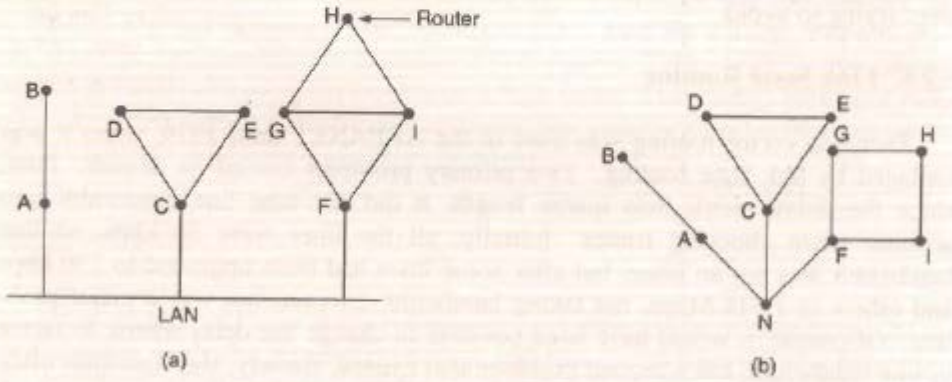


Fig. 5-13. (a) Nine routers and a LAN. (b) A graph model of (a).

One way to model the LAN is to consider it as a node itself, as shown in Fig. 5-13(b). Here we have introduced a new, artificial node, *N*, to which *A*, *C*, and *F* are connected. The fact that it is possible to go from *A* to *C* on the LAN is represented by the path *ANC* here.

Measuring Line Cost

The link state routing algorithm requires each router to know, or at least have a reasonable estimate, of the delay to each of its neighbors. The most direct way to determine this delay is to send a special ECHO packet over the line that the other

side is required to send back immediately. By measuring the round-trip time and dividing it by two, the sending router can get a reasonable estimate of the delay. For even better results, the test can be conducted several times, and the average used.

An interesting issue is whether or not to take the load into account when measuring the delay. To factor the load in, the round-trip timer must be started when the ECHO packet is queued. To ignore the load, the timer should be started when the ECHO packet reaches the front of the queue.

Arguments can be made both ways. Including traffic-induced delays in the measurements means that when a router has a choice between two lines with the same bandwidth, one of which is heavily loaded all the time and one of which is not, it will regard the route over the unloaded line as a shorter path. This choice will result in better performance.

Unfortunately, there is also an argument against including the load in the delay calculation. Consider the subnet of Fig. 5-14, which is divided up into two parts, East and West, connected by two lines, *CF* and *EI*. Suppose that most of the traffic between East and West is using line *CF*, and as a result, this line is heavily loaded with long delays. Including queueing delay in the shortest path calculation will make *EI* more attractive. After the new routing tables have been installed, most of the East-West traffic will now go over *EI*, overloading this line. Consequently, in the next update, *CF* will appear to be the shortest path. As a result, the routing tables may oscillate wildly, leading to erratic routing and many potential problems. If load is ignored and only bandwidth is considered, this problem does not occur. Alternatively, the load can be spread over both lines, but this solution does not fully utilize the best path.

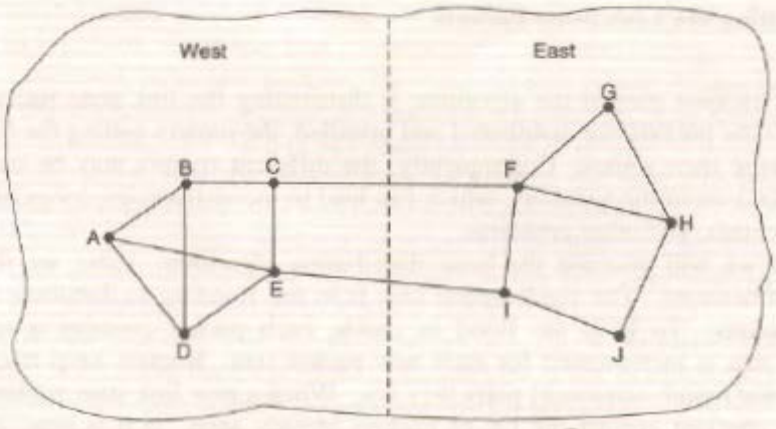


Fig. 5-14. A subnet in which the East and West parts are connected by two lines.

Building Link State Packets

Once the information needed for the exchange has been collected, the next step is for each router to build a packet containing all the data. The packet starts with the identity of the sender, followed by a sequence number and age (to be described later), and a list of neighbors. For each neighbor, the delay to that neighbor is given. An example subnet is given in Fig. 5-15(a) with delays shown in the lines. The corresponding link state packets for all six routers are shown in Fig. 5-15(b).

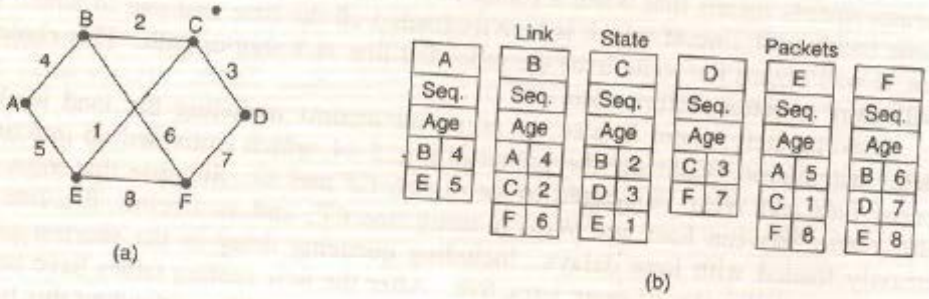


Fig. 5-15. (a) A subnet. (b) The link state packets for this subnet.

Building the link state packets is easy. The hard part is determining when to build them. One possibility is to build them periodically, that is, at regular intervals. Another possibility is when some significant event occurs, such as a line or neighbor going down or coming back up again, or changing its properties appreciably.

Distributing the Link State Packets

The trickiest part of the algorithm is distributing the link state packets reliably. As the packets are distributed and installed, the routers getting the first ones will change their routes. Consequently, the different routers may be using different versions of the topology, which can lead to inconsistencies, loops, unreachable machines, and other problems.

First we will describe the basic distribution algorithm. Later we will give some refinements. The fundamental idea is to use flooding to distribute the link state packets. To keep the flood in check, each packet contains a sequence number that is incremented for each new packet sent. Routers keep track of all the (source router, sequence) pairs they see. When a new link state packet comes in, it is checked against the list of packets already seen. If it is new, it is forwarded on all lines except the one it arrived on. If it is a duplicate, it is discarded.

If a packet with a sequence number lower than the highest one seen so far ever arrives, it is rejected as being obsolete.

This algorithm has a few problems, but they are manageable. First, if the sequence numbers wrap around, confusion will reign. The solution here is to use a 32-bit sequence number. With one link state packet per second, it would take 137 years to wrap around, so this possibility can be ignored.

Second, if a router ever crashes, it will lose track of its sequence number. If it starts again at 0, the next packet will be rejected as a duplicate.

Third, if a sequence number is ever corrupted and 65,540 is received instead of 4 (a 1-bit error), packets 5 through 65,540 will be rejected as obsolete, since the current sequence number is thought to be 65,540.

The solution to all these problems is to include the age of each packet after the sequence number and decrement it once per second. When the age hits zero, the information from that router is discarded. Normally, a new packet comes in, say, every 10 minutes, so router information only times out when a router is down (or six consecutive packets have been lost, an unlikely event). The age field is also decremented by each router during the initial flooding process, to make sure no packet can get lost and live for an indefinite period of time (a packet whose age is zero is discarded).

Some refinements to this algorithm make it more robust. When a link state packet comes in to a router for flooding, it is not queued for transmission immediately. Instead it is put in a holding area to wait a short while first. If another link state packet from the same source comes in before it is transmitted, their sequence numbers are compared. If they are equal, the duplicate is discarded. If they are different, the older one is thrown out. To guard against errors on the router-router lines, all link state packets are acknowledged. When a line goes idle, the holding area is scanned in round robin order to select a packet or acknowledgement to send.

The data structure used by router *B* for the subnet shown in Fig. 5-15(a) is depicted in Fig. 5-16. Each row here corresponds to a recently arrived, but as yet not fully processed, link state packet. The table records where the packet originated, its sequence number and age, and the data. In addition, there are send and acknowledgement flags for each of *B*'s three lines (to *A*, *C*, and *F*, respectively). The send flags mean that the packet must be sent on the indicated line. The acknowledgement flags mean that it must be acknowledged there.

In Fig. 5-16, the link state packet from *A* arrived directly, so it must be sent to *C* and *F* and acknowledged to *A*, as indicated by the flag bits. Similarly, the packet from *F* has to be forwarded to *A* and *C* and acknowledged to *F*.

However, the situation with the third packet, from *E*, is different. It arrived twice, once via *EAB* and once via *EFB*. Consequently, it has to be sent only to *C*, but acknowledged to both *A* and *F*, as indicated by the bits.

If a duplicate arrives while the original is still in the buffer, bits have to be changed. For example, if a copy of *C*'s state arrives from *F* before the fourth

Source	Seq.	Age	Send flags A	Send flags C	Send flags F	ACK flags A	ACK flags C	ACK flags F	Data
A	21	60	0	1	1	1	0	0	
F	21	60	1	1	0	0	0	1	
E	21	59	0	1	0	1	0	1	
C	20	60	1	0	1	0	1	0	
D	21	59	1	0	0	0	1	1	

Fig. 5-16. The packet buffer for router *B* in Fig. 5-15.

entry in the table has been forwarded, the six bits will be changed to 100011 to indicate that the packet must be acknowledged to *F* but not sent there.

Computing the New Routes

Once a router has accumulated a full set of link state packets, it can construct the entire subnet graph because every link is represented. Every link is, in fact, represented twice, once for each direction. The two values can be averaged or used separately.

Now Dijkstra's algorithm can be run locally to construct the shortest path to all possible destinations. The results of this algorithm can be installed in the routing tables, and normal operation resumed.

For a subnet with n routers, each of which has k neighbors, the memory required to store the input data is proportional to kn. For large subnets, this can be a problem. Also, the computation time can be an issue. Nevertheless, in many practical situations, link state routing works well.

However, problems with the hardware or software can wreak havoc with this algorithm (also with other ones). For example, if a router claims to have a line it does not have, or forgets a line it does have, the subnet graph will be incorrect. If a router fails to forward packets, or corrupts them while forwarding them, trouble will arise. Finally, if it runs out of memory or does the routing calculation wrong, bad things will happen. As the subnet grows into the range of tens or hundreds of thousands of nodes, the probability of some router failing occasionally becomes nonnegligible. The trick is to try to arrange to limit the damage when the inevitable happens. Perlman (1988) discusses these problems and their solutions in detail.

Link state routing is widely used in actual networks, so a few words about some example protocols using it are in order. The OSPF protocol, which is

increasingly being used in the Internet, uses a link state algorithm. We will describe OSPF in Sec. 5.5.5.

Another important link state protocol is **IS-IS** (**Intermediate System-Intermediate System**), which was designed for DECnet and later adopted by ISO for use with its connectionless network layer protocol, CLNP. Since then it has been modified to handle other protocols as well, most notably, IP. IS-IS is used in numerous Internet backbones (including the old NSFNET backbone), and in some digital cellular systems such as CDPD. Novell NetWare uses a minor variant of IS-IS (NLSP) for routing IPX packets.

Basically IS-IS distributes a picture of the router topology, from which the shortest paths are computed. Each router announces, in its link state information, which network layer addresses it can reach directly. These addresses can be IP, IPX, AppleTalk, or any other addresses. IS-IS can even support multiple network layer protocols at the same time.

Many of the innovations designed for IS-IS were adopted by OSPF (OSPF was designed several years after IS-IS). These include a self-stabilizing method of flooding link state updates, the concept of a designated router on a LAN, and the method of computing and supporting path splitting and multiple metrics. As a consequence, there is very little difference between IS-IS and OSPF. The most important difference is that IS-IS is encoded in such a way that it is easy and natural to simultaneously carry information about multiple network layer protocols, a feature OSPF does not have. This advantage is especially valuable in large multiprotocol environments.

Q.7 a. Consider that you have two browser applications which are active at the same time, and suppose that your two active applications are accessing the same server to retrieve HTTP documents at the same time. Explain how does the server will differentiates between the two applications? (4)

Answer:

A client application generates an ephemeral port number for every TCP connection it sets up. An HTTP request connection is uniquely specified by (TCP, client IP address, ephemeral port #, server IP address, 80). Both applications in the above situations will have different ephemeral port #s and hence distinguishable to the server.

b. Using RSA, choose p = 3 and q = 11, and encode the phrase "hello". Apply the decryption algorithm to the encrypted version to recover the original plaintext message. (9)

Answer:

We are given $p = 3$ and $q = 11$. We thus have $n = 33$ and $q = 11$. Choose $e = 9$ (it might be a good idea to give students a hint that 9 is a good value to choose, since the resulting calculations are less likely to run into numerical stability problems than other choices for e.) since 3 and $(p-1)*(q-1) = 20$ have no common factors. Choose $d = 9$ also so that $e*d = 81$ and thus $e*d-1 = 80$ is exactly divisible by 20. We can now perform the RSA encryption and decryption using $n = 33$, $e = 9$ and $d = 9$.

letter	m	m**e	ciphertext = m**e mod 33
h	8	134217728	29
e	5	1953125	20
l	12	5159780352	12
l	12	5159780352	12
o	15	38443359375	3

```
ciphertext    c**d              m = c**d mod n   letter
29           14507145975869     8                h
20           512000000000       5                e
12           5159780352         12               l
12           5159780352         12               l
3            19683              15               o
```

c. A router has the following (CIDR) entries in its routing table:

Address/mask	Next hop
128.114.56.0/22	Interface 0
128.114.60.0/22	Interface 1
192.168.30/23	Router 1
Default	Router 2

For packets with the following IP addresses, show where the router will send the packet: (5)

(i) 128.114.52.02 **(ii) 128.114.63.09**

(iii) 192.168.33.05 **(iv) 128.114.57.11**

Answer: ((i) Router 2 (ii) Interface 1 (iii) Router 2 (iv) Interface 0

TEXT BOOK

I. Leon Garcia and Indra Widjaja, Communication Networks: Fundamental Concepts and Key Architecture, 2nd ed., Tata McGraw-Hill, 2004.