Dhirubhai Ambani Institute of Information & Communication Technology
Mid Semester Test-2, 1st Semester 2017- 18

Course Title	IT304 Computer Networks		**Max Marks**	25
Date	12 October 2017	**CLOSED BOOK**	**Time**	90 min

---

1. In the context of a reliable link-layer protocol
   a. Define the efficiency of the protocol.
   b. Compute the efficiency of GBN and SR for a noisy link with constant RTT.
   c. What is the optimal window size and maximum sequence number for each of these protocols? [1+4+1]

2. A transport layer congestion control protocol like TCP relies only on the information contained in ACK feedback received from the receiver along with estimating packet timeout timer.
   a. Describe the TCP timeout timer estimation for each packet.
   b. Describe how the packet timeout may be used to manage congestion.
   c. Describe how the RTT values of the ACK may be used to manage congestion.
   d. Under appropriate assumptions, show that the dependency of expected throughput on **RTT** and packet loss probability **p**. [2+2+2+3]

3. A node can use an *echo* packet to find the round-trip-delay to its direct neighbors. Let's define the weight of a link between neighbor nodes A and B as $W_{AB} = W_{BA} = (RTT_{AB} + RTT_{BA})/2$.
   a. Design this *echo* protocol to compute W_{AB}.
   b. Design a protocol that uses repeated communication between direct neighbors to find the shortest path between any pair of nodes in the network. [2+3]

4. A UDP socket application typically needs to build flow-control logic within the application itself. This requires, among other things, implementing sliding window operations (buffer management, packet sequencing), and retransmissions.
   Using a combination of socket library functions and system calls, provide a pseudocode skeleton for implementing these. Detailed code is not required but familiarity with the function and system calls signature and their usage is expected. [5]

Q1

(a) Efficiency η = $\frac{\text{time spent transmitting useful data}}{\text{total time spent (incl. retrans. \& wait)}}$ [1]

(b) GBN: If $W \cdot T_{trans} \geq (T_{trans} + 2T_{prop})$

there is no waiting for ACK; $W = W_{opt}$

$W_{opt} = 1 + 2a$ [2]

In GBN, every timeout leads to the whole window retransmitted

$$\langle N_r \rangle = \sum_{i=1}^{\infty} [1 + (i-1)W] p^{i-1}(1-p) = \frac{(1-p+Wp)}{(1-p)}$$

$$\eta_{GBN} = \frac{1}{\langle N_r \rangle} \text{ (with optimal Window)} = \boxed{\frac{1-p}{1+2ap}}$$

[2] SR $\eta_{SR} = (1-p)\,\text{Min}\left(1, \frac{W}{W_{opt}}\right) = \boxed{(1-p)}$ for opt. window.

(c) GBN: Max Seq. Size $\geq$ Window size + 1 [1]

SR: Max Seq. Size $\geq$ 2 * Window size.

Q2.

(a) $\langle RTT \rangle^{new} = (1-\alpha)\langle RTT \rangle^{old} + \alpha \cdot RTT^{sample}$; $0 < \alpha < 1$

Suggested $\alpha = 1/8$. $\beta = \frac{1}{4}$

2 $\langle devRTT \rangle^{new} = (1-\beta)\langle devRTT \rangle^{old} + \beta \cdot |RTT^{sample} - \langle RTT \rangle|$

$Timeout = \langle RTT \rangle + 4 * \langle devRTT \rangle$

(b) Packets timeout either because it got corrupted or the Ack took lot longer to arrive. If it is assumed that packet corruption probability is quite low (ignorable),

2 timeout ⇒ large queueing delay & congestion). Hence window size can be reduced at timeout (multiplicative decrease) to manage congestion.

(c) Relatively low RTT value is an indicator of less congested network. This may be used to adjust rate before

2 congestion related packet loss occurs. Here, rate is linearly increased/decreased as RTT values becomes small/large in relation to RTT_{min} observed over the session.

(d) $\#\text{ of pkts} = \left(\frac{3W}{4}\right)\cdot\left(\frac{T}{RTT}\right) = \frac{1}{p}$.

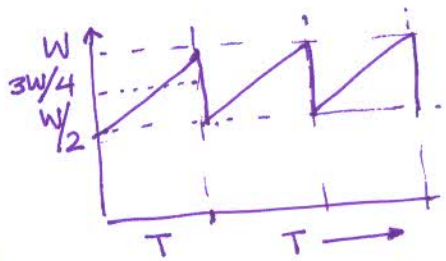
Also Window increases by 1 every RTT

3 $\Rightarrow T = \frac{W}{2} \cdot RTT \Rightarrow \frac{T}{RTT} \propto W$

$\Rightarrow W^2 \propto \frac{1}{p}$.

$\langle X \rangle = \frac{\#\text{ of pkts in a cycle}}{\text{Time}} = \frac{1/p}{RTT \cdot \frac{W}{2}}$

$\Rightarrow \boxed{\langle x \rangle \propto \frac{1}{RTT \cdot \sqrt{p}}}$



- Ignore slow-start phase
- Assume pkt loss happens exactly every $\frac{1}{p}$ pkts.
- $\langle X \rangle = \frac{\langle W \rangle}{\langle RTT \rangle}$.

Ques 3 .A node can use an *echo* packet to find the round-trip-delay to its direct neighbors. Let's define the weight of a link between neighbor nodes A and B as $W_{AB} = W_{BA} = (RTT_{AB} + RTT_{BA})/2$. (2+3)

a. Design this *echo* protocol to compute W_{AB}. (1+1)

1. Learning about its 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/ECHO packet on each point-to-point line. The router on the other end is expected to send back a reply to the query.

   Hello packet structure:

Source Ip address	Source port no.	Destination port no.	"Hello"	Timer

2. Measuring Line Cost: To estimate of the delay to each of its neighbors. The most direct way to determine this delay is to send over the line a special ECHO packet that the other side is required to send back immediately.

   - Start the timer and flag =0 for echo packet from A to B.
   - Send a packet with the following structure from A to B :

Source Ip Address	Destination Ip Address	Timer	ACK	Flag

   - Upon receiving the packet at B, it store the value of timer and set the flag =1 in order to remove any packet loss .
   - Send the Same Echo packet as response from B to A in the following form:

Source Ip Address	Destination Ip Address	Timer	ACK	Flag

   - Upon receiving the packet with flag =1, stop the timer and store the value of RTT_{AB} = Timer.

   Similarly we will perform follow same procedure for calculating RTT_{BA} from B to A.

   - By measuring the round trip time as RTT_{AB} and RTT_{BA} then summing both and then dividing it by two, we will get the average W_{AB} OR W_{BA} .
     For even better results, the test can be conducted several times, and the average used.

$W_{ab} = W_{ba} = (RTT_{ab} + RTT_{ba})/2$

b. Design a protocol that uses repeated communication between direct neighbors to find the shortest path between any pair of nodes in the network. (1+1+1)

In order to compute shortest path between any pair of nodes in network we can use Distance Vector routing protocol or Bellman Ford Algorithm.

A distance-vector routing protocol requires that a router inform its neighbours of topology changes periodically due to which it have less computational complexity and message overhead.

Routers using distance-vector protocol do not have knowledge of the entire path to a destination. Instead they use two methods:

1. Direction in which router or exit interface a packet should be forwarded.
2. Distance from its destination

Distance-vector protocols are based on calculating the direction and distance to any link in a network. Here in this answer we are

Define distances at each node X :: dx(y) = cost in form Weight W_{ij} path from X to Y

Update distances based on neighbors :: dx(y) = min {W(x,v) + dv(y)} over all neighbors V.

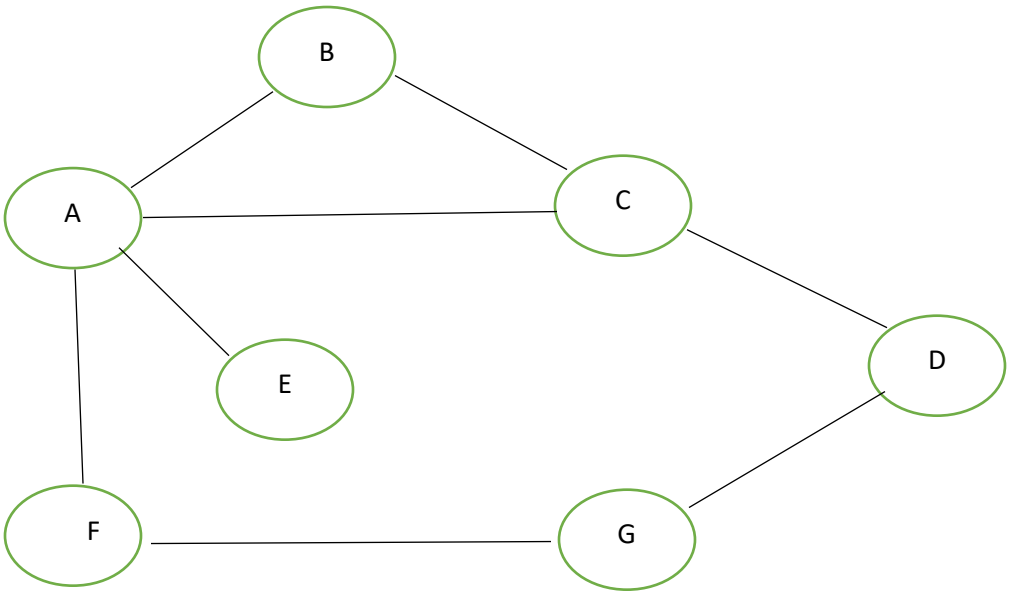
The least cost route between any two nodes is the route with minimum distance.

Algorithm With Example:

Each node constructs a one-dimensional array containing the "distances"(W_{ab}) to all other nodes and distributes that vector to its immediate neighbors.

1. The starting assumption for distance-vector routing is that each node knows the cost of the link to each of its directly connected neighbors.
2. A link that is down is assigned an infinite cost.

Each node maintains a vector (table) of minimum distance to every node. The cost of reaching a destination is calculated using various route metrics.



Information Stored at Node	Distance to Reach Node						
	A	B	C	D	E	F	G
A	0	1	1	?	1	1	?
B	1	0	1	?	?	?	?
C	1	1	0	1	?	?	?
D	?	?	1	0	?	?	1
E	1	?	?	?	0	?	?
F	1	?	?	?	?	0	1
G	?	?	?	1	?	1	0

? MEANS INFINITY

Table shows the initial distances stored at each node

We can represent each node's knowledge about the Weights to all other nodes as a table like the one given in above table

Note that each node only knows the information in one row of the table.

1. Every node sends a message to its directly connected neighbors containing its personal list of Weights (W). (for example, **A** sends its information to its neighbors **B,C,E**, and **F**.)
2. If any of the recipients of the information from **A** find that **A** is advertising a path shorter than the one they currently know about, they update their list to give the new path length and note that they should send packets for that destination through **A**. (node **B** learns from **A** that node **E** can be reached at a cost of 1; **B** also knows it can reach **A** at a cost of 1, so it adds these to get the cost of reaching **E** by means of **A**. **B** records that it can reach **E** at a cost of 2 by going through **A**.)
3. After every node has exchanged a few updates with its directly connected neighbors, all nodes will know the least-cost path to all the other nodes.
4. In addition to updating their list of Cost in terms of weights when they receive updates, the nodes need to keep track of which node told them about the path that they used to calculate the cost, so that they can create their forwarding table.
5. For example, **B** knows that it was **A** who said " I can reach **E** in one W_{AE}" and so **B** puts an entry in its table that says " To reach **E**, use the link to **A**.

Information Stored at Node	Distance to Reach Node						
	A	B	C	D	E	F	G
A	0	1	1	2	1	1	2
B	1	0	1	2	2	2	3
C	1	1	0	1	2	2	2
D	2	2	1	0	3	2	1
E	1	2	2	3	0	2	3
F	1	2	2	2	2	0	1
G	2	3	2	1	3	1	0

Table 2 Final distance stored at each node

Each node's forwarding table consists of a set of triples of the form: (Destination, Cost(W_{ij}), NextHop).

For example, Table 3 shows the complete routing table maintained at node B for the network in figure1.

Destination	Cost	NextHop
A	1	A
C	1	C
D	2	C
E	2	A
F	2	A
G	3	A

Table 3: Routing table maintained at node B

NOTE with Marks Distribution:

For Q3 a part

Explanation with Message Structure + W_{AB} derivation then 1+1

For Q3 b part

Marks have not been given for using link state protocol or link state using flooding as well as using Dijkstra's Algorithm with flooding in order to find shortest path.

Besides it if you have used any other Algorithm or protocol but have not used above W_{ij} as Weighted cost then also marks have been deducted.

Algorithm + Example + Proper Table Structure then 1+1+1

Q4. *A UDP socket application typically needs to build flow-control logic within the application itself. This requires, among other things, implementing sliding window operations (buffer management, packet sequencing), and retransmissions. Using a combination of socket library functions and system calls, provide a pseudo code skeleton for implementing these. Detailed code is not required but familiarity with the function and system calls signature and their usage is expected.*

Answer should contain following components:

- Basic Socket library functions for initial connection and data exchange.
- Given Window Size (W) and Sequence Number, what type of Data structure for Buffer management you are going to use.
- RTT Estimation to calculate Timeout for retransmission of packet.
- System calls for implementing Timer.
- How you manage Window (how Sliding of window occurs.) and brief explanation of algorithm you used for sliding window (i.e., GBN or SR).

1).Basic Socket library functions for initial connection and data exchange.

```
1..socket (domain, type, protocol)

2..bind(udpSocket, (struct sockaddr *) &serverAddr, sizeof(serverAddr));

3..send to(int socket, const void *buffer, size_t length, int flags, const
struct sockaddr *dest_addr, socklen_t dest_len)

4..recvfrom(int socket, void *restrict buffer, size_t length, int flags,
struct sockaddr *restrict src_addr,socklen_t *restrict *src_len)
```

Exact syntax not required but socket library functions for UDP should be mentioned.

Data Structure -How you manage Buffer

If size of each packet is constant then circular array (else an appropriate DS) can be used as a buffer(Data Structure). Pointers(ACK and OUTSTANDING) are maintained to keep track of ACKed packets and the outstanding packets. Load the buffer by replacing ACKed packets with new packets.

- If SW protocol is SR(Selective Repeat) then timer is maintained for each packet .
- If GBN is used then Timer is maintained for the packet sent first or the one pointed by ACK pointer.
- What happens when Sequence numbers exhausted.

RTT Estimation for time out:

(Store current time after transmitting the packet and subtract it from the time when its ACK arrives.)

You need a **gettimeofday()** or similar system call to get current time with millisecond accuracy.Formula for timeout is that given in the solutions of Q2.Here, Exact equations are not required but idea should be explained briefly.

System calls for implementing Timer. (syntax not required)

1).poll(): `int poll(struct pollfd fds[], nfds_t nfds, int timeout);`

2).select(): `int select(int nfds, fd_set *readfds, fd_set *writefds,`

`fd_set *errorfds, struct timeval *timeout);`

For Selective Repeat, poll() should be called for each packet in window and for GBN poll() should be called for First packet pointed by ACK pointer. When poll() returns 0 ,all the packets between ACK and OUTSTANDING pointers should be retransmitted.

How you manage Window (how Sliding of window occurs.) and brief explanation(pseudo code) of algorithm you used for sliding window (i.e., GBN or SR).

Ex. For SR

```
While (true)
        While( ((ACK-OUTSTANDING+1)%W)<MAX_OUTSTANDING_ALLOWED)
                Transmit_pkt(OUTSTANDING);
                Increment OUTSTANDING
                Insert_timer();//select() or poll()
        End
End
```

Here, ACK pointer increments when sender receives Acknowledgement

ACKed	Pkt-1	Pkt-2	Pkt-3	Pkt-4	NULL	NULL

^ ^

ACK OUTSTANDING