ASSIGNMENT 7

Implementing a Reliable Transport Protocol
using Datagram Socket

Arijit Panigrahy | 14CS30005

Aniket Choudhary | 14CS30004

04.04.2017

Computer Networks Laboratory

OBJECTIVE

To implement a reliable transmission protocol on top of a datagram socket. Also, we have to implement flow control in our protocol

STEPS FOLLOWED

This assignment consists of 6 major parts (or stages) as follows :-

- gbn.h
  In this part, we have to mention the prototypes of the functions and all the structs, so that it can be used in other files. Here, the packet_header, socket_state, timeout values, window size etc have been defined.
- sender.c
  This is just a test file that invokes the functions from gbn.h and gbn.c to transfer a file. We first open a socket, then connect to the server. After that, we read from the file and transfer it to the client. After the transfer is over, we simply close the socket. Here, we have to keep track of number of packets sent. Time values are also noted for calculating throughput.
- receiver.c
  This is just a test file that invokes the functions from gbn.h and gbn.c to receive a file. We first open a socket, bind it to the port and listen for connections. Once, we find one, we accept it and receive the file. At the end, we close the socket. Here, we have to keep track of number of packets received. Time values are also noted for calculating throughput.
- gbn.c
  This file implements the following functions:-
  - int gbn_connect(int sockfd, const struct sockaddr *server, socklen_t socklen);
    This fuction is used to check connection using 3 way handshaking. It sends a SYN packet, waits for SYNACK and then sends DATAACK
  - void fill_header(uint16_t type, uint16_t seqnum, gbnhdr *hdr)
    It is used to initialize the packet with given type and seqnum.
  - int gbn_listen(int sockfd, int backlog);
    It increments count (global) and allows only backlog clients to access.

  - int gbn_bind(int sockfd, const struct sockaddr *server, socklen_t socklen);
    Just calls bind().
  - int gbn_socket(int domain, int type, int protocol);

Initializes the socket_state and calls socket().

- int gbn_accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
  It waits for receiving SYN and then sends SYNACK and again waits for DATAACK. Once received, it returns success.
- int gbn_close(int sockfd);
  Calls close(sockfd).
- void type(void * buf);
  Returns the type of the packet (i.e.SYN/SYNACK/DATA…)
- ssize_t gbn_send(int sockfd, const void *buf, size_t len, int flags);
  It implements the Go back N algorithm. It sends 'w' packets and waits for acknowledgement after setting the alarm of timeout 't'. If it receives a packet within the timeout, we increment free_window by (ack_seq-last_ack_seq) and send some more packets and reset alarm else, we resend all the packets since last_ack_seq.
- ssize_t gbn_recv(int sockfd, void *buf, size_t len, int flags)
  The expected_seq is set to 3 (1 for SYNACK and DATAACK each). Once, we receive a packet within a timeout 't', we check if its seq is same as the expected_seq. i f yes, we update the expected_seq and send an acknowledgment else we do nothing. If the timeout expires, we quit.
- ssize_t send_packet(int s, const void *buf, size_t len, int flags,const struct sockaddr *to, socklen_t tolen)
  It just invokes sendto().

- udp_client.c
  This module sends a file using DATAGRAM SOCKET.
- udp_server.c
  This module receives the file using DATAGRAM SOCKET.
- Topo.py
  A simple topo having h1 and h2 with a link between them.
- Makefile
  It is written to compile the code.
- Testing the gbn.c
  In order to run the topo and transfer the file using gbn (i) and udp(ii), we have follow the following steps:-
  1. Open terminal and ssh to mininet.
  2. Run the topo using
     Sudo mn --custom topo.py --topo mytopo--link tc

3. Open Xterms using,
   xterm h1 h2
4. Make two directories h1, h2 and place the files (gbn.c, gbn.h, gbn_sender.c, gbn_receiver.c, makefile, udp_sender.c, udp_receiver.c) in both directory and make it.
   Mkdir h1
   Mkdir h2
   Make
5. In xterm of h1
   Let it be the sender.
   If you want to use go back N, use
   ./gbn_sender port_no file_name
   Else
   ./udp_client
6. In xterm of h2
   Let it be the receiver.
   If you want to use go back N, use
   ./gbn_receiver port_no file_name
   Else
   ./udp_server
7. Run tcpdump or wireshark and take observations.

Make sure to change the SERVER_IP address in the client files before running.

OBSERVATIONS AND DISCUSSIONS

1) Packet Header that we have used is:
   typedef struct {
   bool type; //SYN/DATA
   bool subtype; //ACK or Normal
   unsigned int seqnum; //Sequences No/Ack No
   *char data[DATALEN]; } //Message
   attribute__((packed)) gbnhdr;

2) Design Details:-
   When you open the socket, we initialize the socket_state parameters.
   s.seq=1 (next sequence to be received).
   s.state=CLOSED.

When you try to connect using gbn_connect,
The state of socket can be SYN_SENT, ESTABLISHED or CLOSED.
When you send the SYN packet, your state is SYN_SENT. Then you wait for the acknowledgement. Upon receiving, you send DATAACK packet and change your state you ESTABLISHED. If connect fails midway, you return and change your state to CLOSED.

When you are accepting using gbn_accept,
The state of the socket can be CLOSED, SYN_RCVD, ESTABLISHED. When your state is CLOSED, you wait for SYN to arrive. Upon receiving, you change the state to SYN_RCVD and send the SYNACK. When the DATAACK arrives, the state is changed to ESTABLISHED.
If accept fails midway, you return and change your state to CLOSED.

When sending using gbn_send,
It implements the Go back N algorithm. It sends 'w' packets and waits for acknowledgement after setting the alarm of timeout 't'. If it receives a packet within the timeout, we increment free_window by (ack_seq-last_ack_seq) and send some more packets and reset alarm else, we resend all the packets since last_ack_seq. Whenever the acknowledgement is received, the s.seqnum is updated accordingly.

When receiving using gbn_recv,
The expected_seq is set to 3 (1 for SYNACK and DATAACK each). Once, we receive a packet within a timeout 't', we check if it's seq is same as the expected_seq. i f yes, we update the expected_seq and send an acknowledgment else we do nothing. If the timeout expires, we quit. Whenever the packet is received, the s.seqnum is updated accordingly.

Rest all functions simply call to established socket system calls.

3) <u>Design Parameter:-</u>
We have taken the following values:-
Packet_Length = 1024 Bytes
Timeout = 10 sec
Window = 6 packets

It can be changed in gbn.h .

Packet_Length: If the packet_length is too small, then the overhead will be large for the

packet header. If it is large, then if a packet is lost => a lot of useful data has to be retransmitted => again waste of resources.
Thus, keeping it at 1 KB seems fine.
Timeout: A large timeout leads to low efficiency since we are doing nothing but waiting. Where as, if timeout is too small, we might resend the packets even before waiting completely for the acknowledgment => wastage of bandwidth again.
Thus, Timeout=10 sec seems fine for our protocol.
Window: Having a smaller window lowers the data transmission speed whereas having a larger window means if a packet is lost, then all those packets have to be sent again, which leads to wastage of bandwidth.
Thus, we decide to keep it close to 10.

4) <u>Performance Measures:-</u>
In case of RelTP, we observe that no packet is lost. But time taken to transfer a small 1 KB file is appx 10 secs. But there is absolutely no packet loss. In case of normal DATAGRAM SOCKET, the time taken is hardly 1 sec, but data loss is huge sometimes ~50%.

The results are shown below in the table and their corresponding plots:-

Bandwidth = 1 Mbps

LOSS	Packet Sent	Bytes Sent	Packet recv	Bytes Received	Server Start	Client End	Time Duration (in ms)	Throughput	Packet loss rate
1	205300	105934800	171001	87552512	382493	4875591	449309.8	779.4400389	0.1670677058
3	205300	105934800	167907	85968384	392477	4876424	448394.7	766.8991984	0.1821383341
5	205300	105934800	162007	82947584	555449	5036366	448091.7	740.4518673	0.2108767657
10	205300	105934800	160132	81987584	127716	4609484	448176.8	731.7432228	0.2200097418
15	205300	105934800	100809	51614208	446029	4930895	448486.6	460.3411384	0.5089673648

Bandwidth = 10Mbps

LOSS	Packet Sent	Packet recv	Server Start	Client End	Time Duration (in ms)
1	205300	172520	693909	1195240	50133.1
3	205300	162021	738586	1236101	49751.5
5	205300	153967	691848	1188974	49712.6
10	205300	155380	246930	744010	49708
15	205300	119217	308554	817594	50904

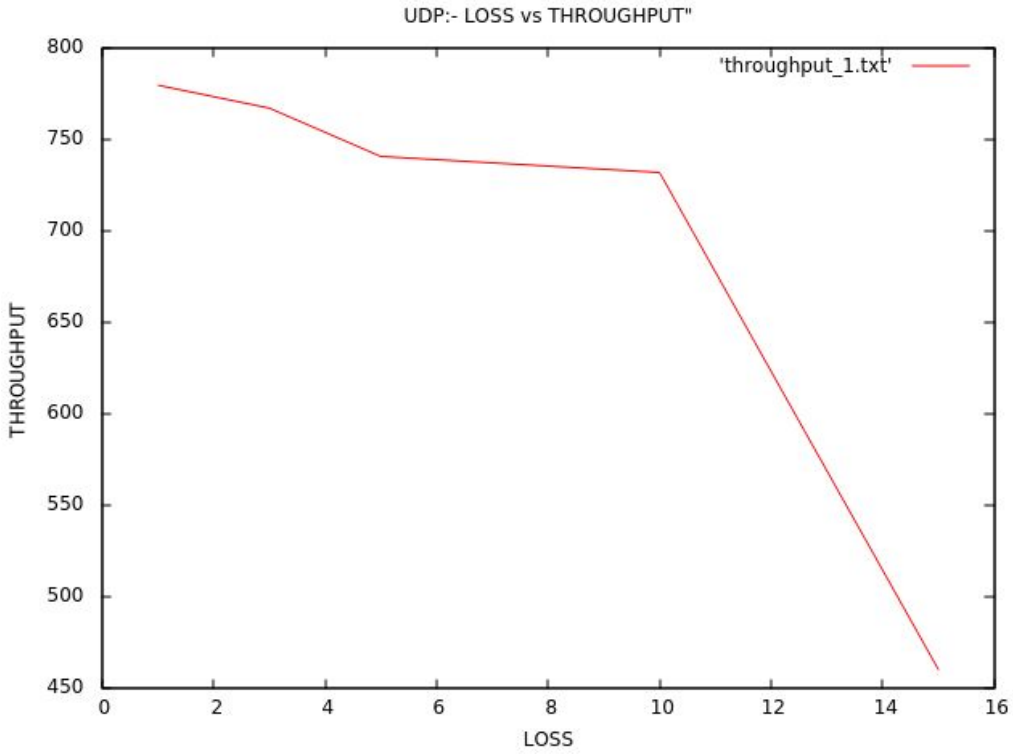
For UDP

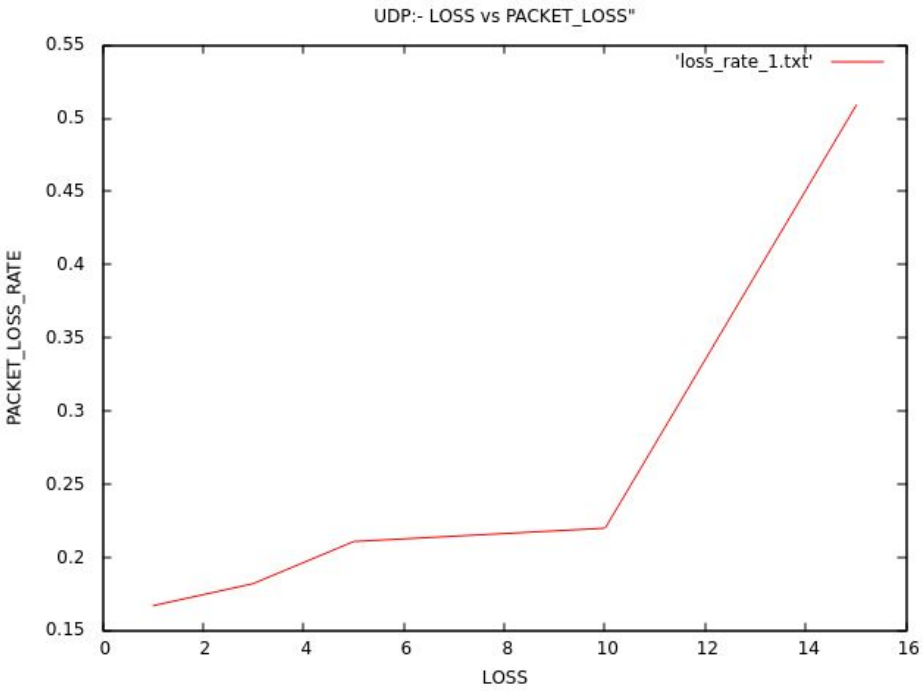
Bandwidth = 1 Mbps						
LOSS	Packets	Server Start	Client End	Time Duration (in s)	Packet Loss	Throughput (in KBps)
1	99201	417125	9912029	949.4904	0	106.9856251
3	97812	198766	15892350	1569.3584	0	63.82193386
5	96524	630270	48912465	4828.2195	0	20.47143383
10	95121	43276	81820347	8177.7071	0	11.91090642
15	95321	37212	91421714	9138.4502	0	10.68110039

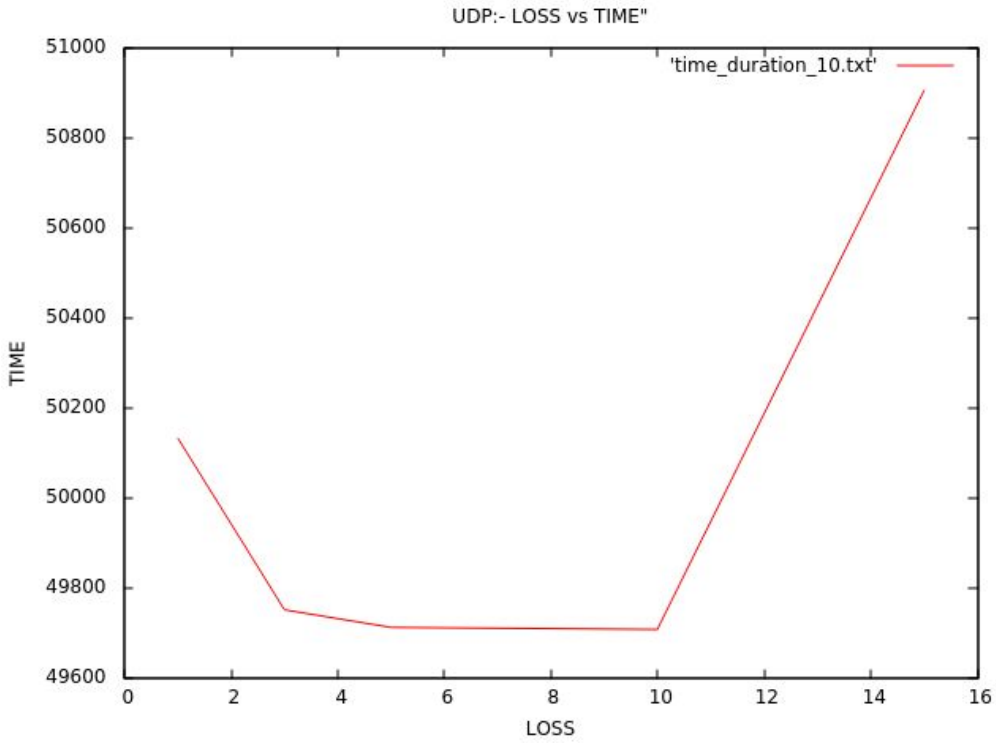
Bandwidth = 10 Mbps					
LOSS	Packets	Server Start	Client End	Time duration (in ms)	Packet Loss
1	99022	418105	9119549	870.1444	0
3	97913	196576	16924215	1672.7639	0
5	96952	628976	49014596	4838.562	0
10	95331	42941	82210211	8216.727	0
15	95411	37541	90842946	9080.5405	0

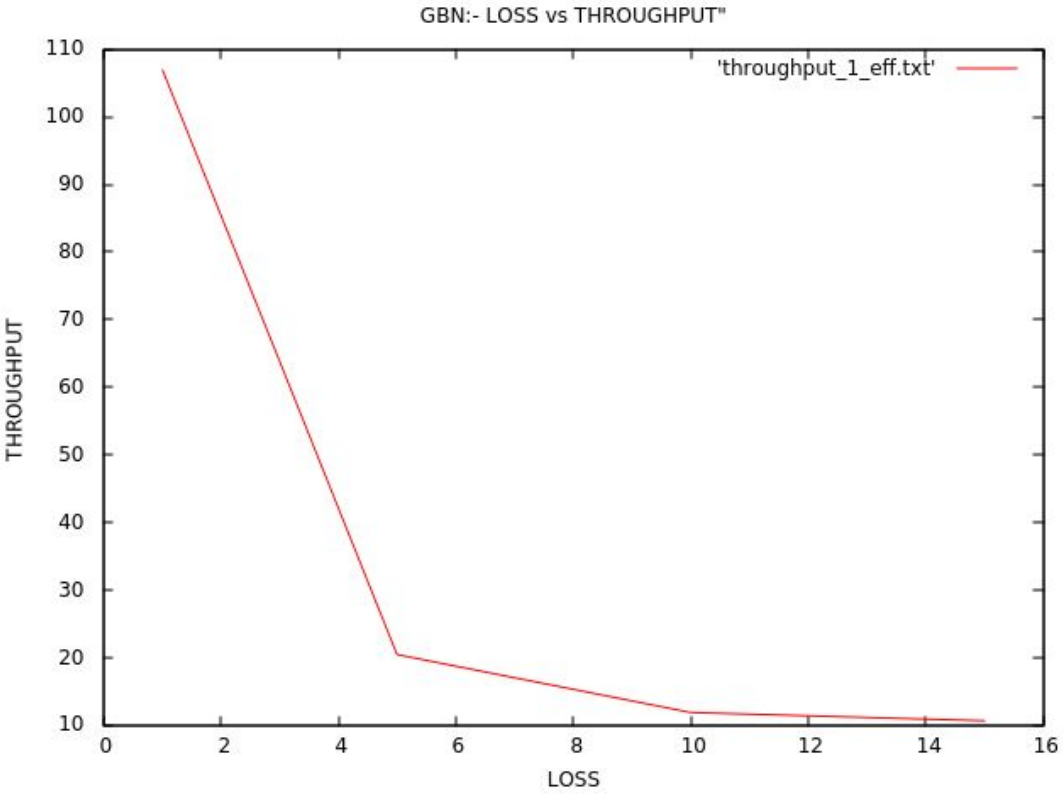
For GO BACK N

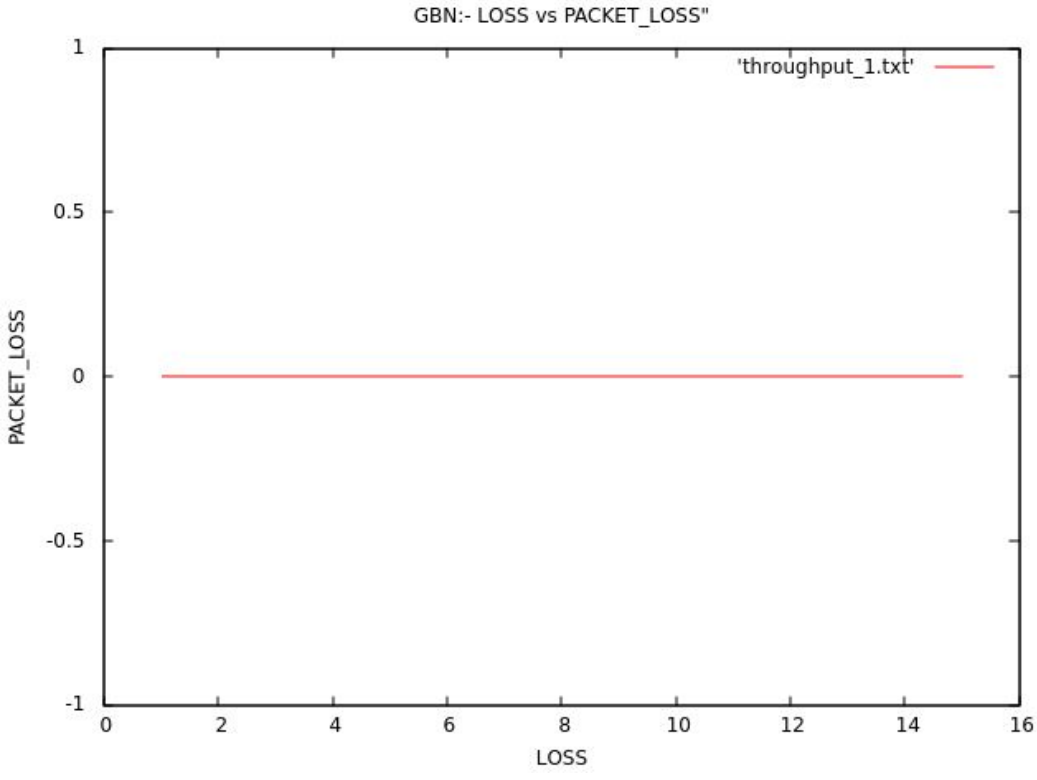
The plots are given below:-

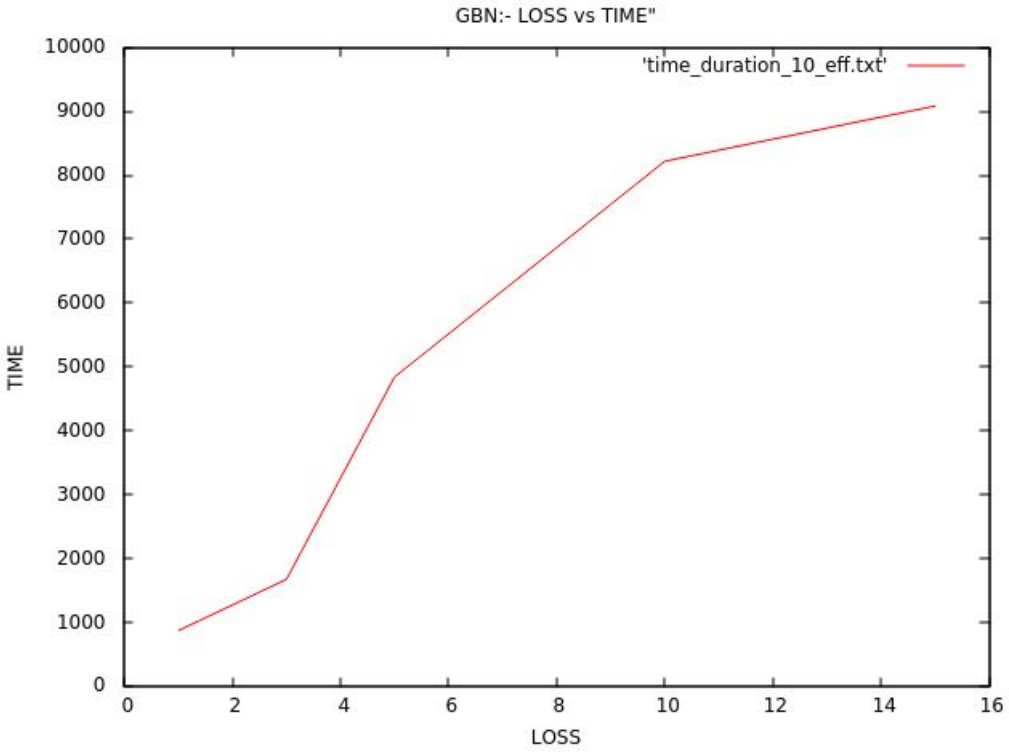



UDP:- LOSS vs PACKET_LOSS"
'loss_rate_1.txt'
PACKET_LOSS_RATE
LOSS
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0
2
4
6
8
10
12
14
16


UDP:- LOSS vs TIME"
'time_duration_10.txt'
TIME
LOSS
51000
50800
50600
50400
50200
50000
49800
49600
0
2
4
6
8
10
12
14
16


GBN:- LOSS vs THROUGHPUT"
'throughput_1_eff.txt'
THROUGHPUT
LOSS
110
100
90
80
70
60
50
40
30
20
10
0
2
4
6
8
10
12
14
16


GBN:- LOSS vs PACKET_LOSS"
'throughput_1.txt'
PACKET_LOSS
LOSS
1
0.5
0
-0.5
-1
0
2
4
6
8
10
12
14
16



As it can be observed from the graphs,

1) TIME TAKEN by GBN is much higher than UDP. It also increases with loss rate. But, TIME TAKEN UDP is not affected by LOSS RATE.
2) PACKET LOSS is 0 for GBN. But for UDP, PACKET LOSS increases with LOSS RATE.
3) THROUGHPUT decreases for both UDP and GBN with increase in LOSS. But the decrease is much more steep in case of GBN rather than UDP.