Programming Assignment 2
Reliable Transport Protocols

I have read and understood the course's academic integrity policy.

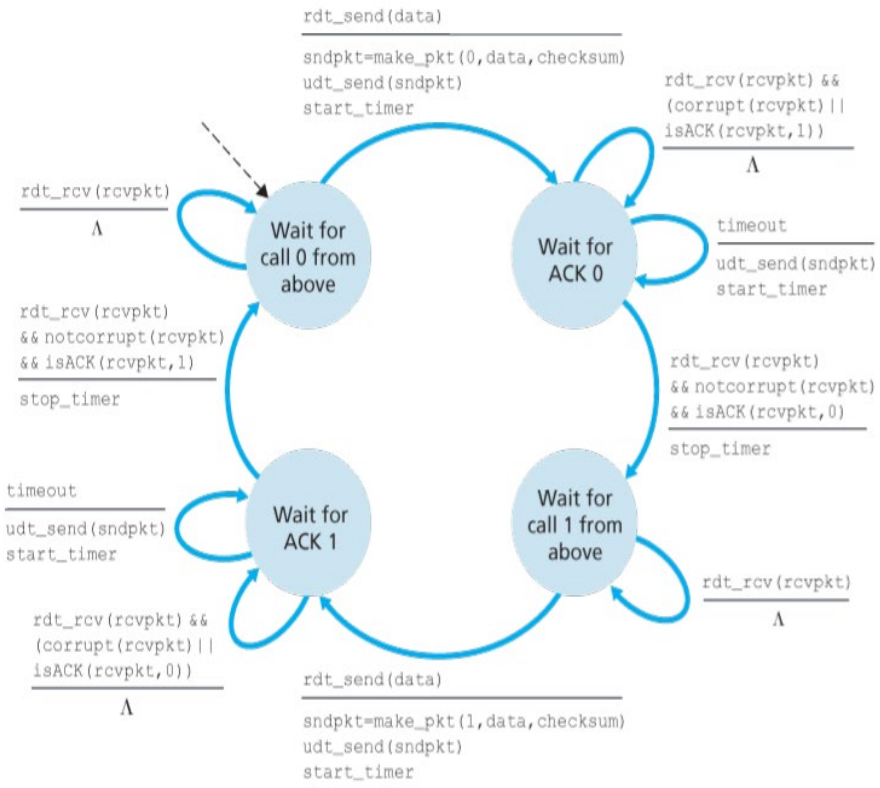
GROUP 18

Group Members & their contributions

- Saqib Hussain Khan
  - UB IT name – saqibhus
  - UB person no. – 50431904
- Sougato Bagchi
  - UB IT name – sougatob
  - UB person no. – 50411918
- Gaurav Toravane
  - UB IT name – gtoravan
  - UB person no. – 50282478

IMPLEMENTATION OF ABT PROTOCOL

- ABT protocol is implemented as per below given FSM description taken from the textbook Computer Networking _ A Top Down Approach, 7th edition.



- A flag **ACK_FLAG** is maintained which represents 1 as ready to transmit new packet and 0 otherwise.
- **ACK_FLAG** initially holds 1 and gets updated to 1 every time we receive a successful acknowledgment from receiver B.
- When **ACK_FLAG** is 0 then any incoming data from layer5 is stored in a buffer of size 1000.

```
struct msg messageBuffer[1000];
```

- When timer interrupts, current packet is transmitted, and timer is started again.
- When successful acknowledgment is received then packet is either transmitted from stored buffer (if not empty) or ACK_FLAG is changed to 1 so that new packet is transmitted whenever received from upper layer5.

```
if(msgNum<=bufferUsed){
  new_packet = (struct pkt){0};
  strncpy(new_packet.payload, messageBuffer[msgNum].data, 20);
  msgNum++;
  if(SEQ_NUM == 0)
    SEQ_NUM = 1;
  else
    SEQ_NUM = 0;
  new_packet.seqnum = SEQ_NUM;
  int checksum = compute_checksum(new_packet);
  new_packet.checksum = checksum;
  tolayer3(0, new_packet);
  starttimer(0, TIMEOUT);
}else{
  ACK_FLAG = 1;
}
```

- Checksum of every packet is computed as per below code:

```
int compute_checksum(struct pkt new_packet){
  int seqnum, acknum;
  int checksum = 0;
  seqnum = new_packet.seqnum;
  checksum = checksum + seqnum;
  acknum = new_packet.acknum;
  checksum = checksum + acknum;
  new_packet.payload[strcspn(new_packet.payload, "\n")] = 0;
  for(int i = 0; i<20; i++)
    checksum = checksum + new_packet.payload[i];

  checksum = ~checksum;
  return checksum;
}
```

- At the receiver side, ACK_NUM variable is maintained which hold 0 or 1 value and gets updated based on the expected packet sequence. It alternates between 0 and 1 whenever an expected packet is received.

TEST RESULTS OF ABT

SANITY

```
PASS!
Testing with MESSAGES:20, LOSS:0.0, CORRUPTION:1.0, ARRIVAL:1000, WINDOW:0 ...
Running simulator [10 Runs] ...
Run#1 [seed=1234] ...  Done!
Run#2 [seed=1111] ...  Done!
Run#3 [seed=2222] ...  Done!
Run#4 [seed=3333] ...  Done!
Run#5 [seed=4444] ...  Done!
Run#6 [seed=5555] ...  Done!
Run#7 [seed=6666] ...  Done!
Run#8 [seed=7777] ...  Done!
Run#9 [seed=8888] ...  Done!
Run#10 [seed=9999] ...  Done!
PASS!
SANITY TESTS:  PASS
highgate {~/cse489589_assignment2/grader} >
```

BASIC

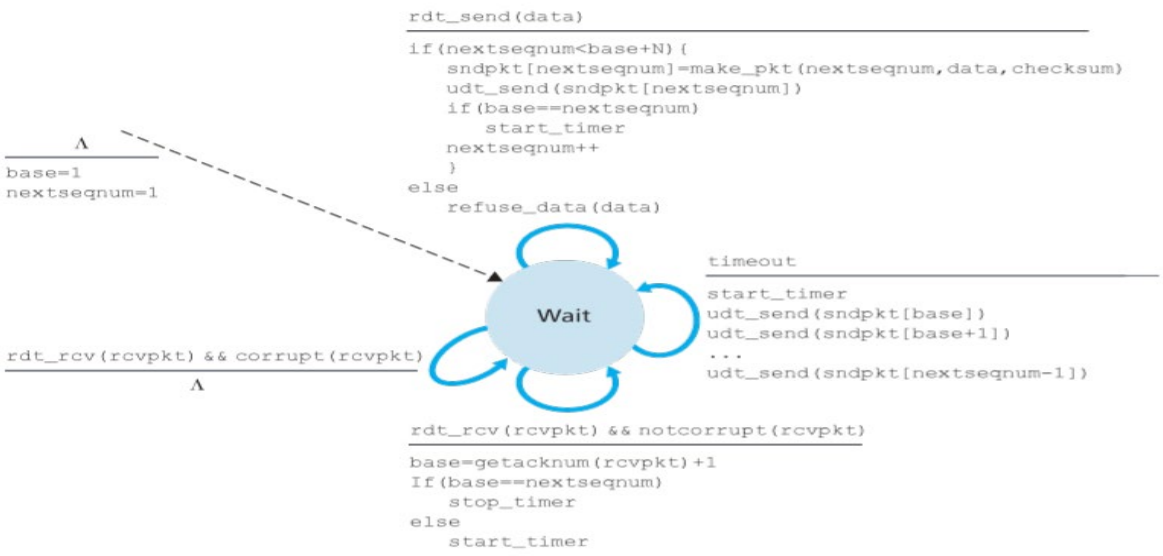
```
Testing with MESSAGES:20, LOSS:0.0, CORRUPTION:0.8, ARRIVAL:1000, WINDOW:0 ...
Running simulator [10 Runs] ...
Run#1 [seed=1234] ...  Done!
Run#2 [seed=1111] ...  Done!
Run#3 [seed=2222] ...  Done!
Run#4 [seed=3333] ...  Done!
Run#5 [seed=4444] ...  Done!
Run#6 [seed=5555] ...  Done!
Run#7 [seed=6666] ...  Done!
Run#8 [seed=7777] ...  Done!
Run#9 [seed=8888] ...  Done!
Run#10 [seed=9999] ...  Done!
PASS!
BASIC TESTS:  PASS
highgate {~/cse489589_assignment2/grader} >
```

ADVANCED

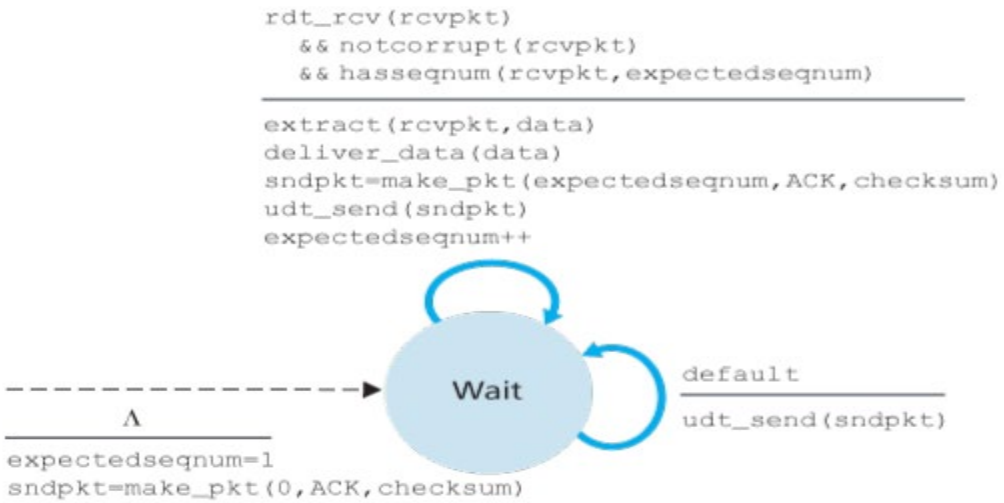
```
Running simulator [10 Runs] ...
Run#1 [seed=1234] ...  Done!
Run#2 [seed=1111] ...  Done!
Run#3 [seed=2222] ...  Done!
Run#4 [seed=3333] ...  Done!
Run#5 [seed=4444] ...  Done!
Run#6 [seed=5555] ...  Done!
Run#7 [seed=6666] ...  Done!
Run#8 [seed=7777] ...  Done!
Run#9 [seed=8888] ...  Done!
Run#10 [seed=9999] ...  Done!
PASS!
ADVANCED TESTS:  PASS
highgate {~/cse489589_assignment2/grader} > ./advanced_tests -p ../saqibhus/abt -r ./run_experiments
```

IMPLEMENTATION OF GBN PROTOCOL

- GBN protocol is implemented as per below given FSM description taken from the textbook Computer Networking _ A Top Down Approach, 7th edition.



Sender A



Receiver B

- GBN protocol is implemented on the concept of window size.
- Packets are transmitted if they lie within the window. If the received packets from layer5 are outside the window size, then such packets are stored in a buffer of size 1000.

```
if(bufferUsed < 1000){
  strcpy(messageBuffer[bufferUsed+1].data, message.data);
  bufferUsed++;
```

- In the below mentioned code segment:

  nextSeqNum : stores the sequence number of next packet to be transmitted
  base : Starting point of the window
  windowSize : Length of the window

```
if(nextSeqNum < base + windowSize){
  new_packet = (struct pkt){0};
  new_packet.seqnum = nextSeqNum;
  strncpy(new_packet.payload, messageBuffer[nextSeqNum].data, 20);
  int checksum = compute_checksum(new_packet);
  new_packet.checksum = checksum;
  tolayer3(0, new_packet);

  if(base == nextSeqNum){
    starttimer(0, TIMEOUT);
  }
  nextSeqNum++;
```

- On timer interrupt, entire window is sent as given in the below code:

```
void A_timerinterrupt()
{
  starttimer(0, TIMEOUT);
  int i = base;
  while(i < nextSeqNum){
    new_packet = (struct pkt){0};
    new_packet.seqnum = i;
    strncpy(new_packet.payload, messageBuffer[i].data, 20);
    int checksum = compute_checksum(new_packet);
    new_packet.checksum = checksum;
    tolayer3(0, new_packet);
    i++;
  }
}
```

Test results of GBN

SANITY

```
Running simulator [10 Runs] ...
Run#1 [seed=1234] ...  Done!
Run#2 [seed=1111] ...  Done!
Run#3 [seed=2222] ...  Done!
Run#4 [seed=3333] ...  Done!
Run#5 [seed=4444] ...  Done!
Run#6 [seed=5555] ...  Done!
Run#7 [seed=6666] ...  Done!
Run#8 [seed=7777] ...  Done!
Run#9 [seed=8888] ...  Done!
Run#10 [seed=9999] ...  Done!
PASS!
SANITY TESTS:  PASS
highgate {~/cse489589_assignment2/grader} > ./sanity_tests -p ../saqibhus/gbn -r ./run_experiments
```

BASIC

```
Running simulator [10 Runs] ...
Run#1 [seed=1234] ...  Done!
Run#2 [seed=1111] ...  Done!
Run#3 [seed=2222] ...  Done!
Run#4 [seed=3333] ...  Done!
Run#5 [seed=4444] ...  Done!
Run#6 [seed=5555] ...  Done!
Run#7 [seed=6666] ...  Done!
Run#8 [seed=7777] ...  Done!
Run#9 [seed=8888] ...  Done!
Run#10 [seed=9999] ...  Done!
PASS!
BASIC TESTS:  PASS
highgate {~/cse489589_assignment2/grader} > ./basic_tests -p ../saqibhus/gbn -r ./run_experiments
```

ADVANCED

```
Running simulator [10 Runs] ...
Run#1 [seed=1234] ...  Done!
Run#2 [seed=1111] ...  Done!
Run#3 [seed=2222] ...  Done!
Run#4 [seed=3333] ...  Done!
Run#5 [seed=4444] ...  Done!
Run#6 [seed=5555] ...  Done!
Run#7 [seed=6666] ...  Done!
Run#8 [seed=7777] ...  Done!
Run#9 [seed=8888] ...  Done!
Run#10 [seed=9999] ...  Done!
PASS!
ADVANCED TESTS:  PASS
highgate {~/cse489589_assignment2/grader} > ./advanced_tests -p ../saqibhus/gbn -r ./run_experiments
```

IMPLEMENTATION OF SR PROTOCOL

multiple software timers in SR using a single hardware timer

- This is implemented by maintaining a queue of unacknowledged packets and their respective start time at which the packet was transmitted to B.

```
float currTime = get_sim_time();
struct timerQueue *new_timer = (struct timerQueue*) malloc(sizeof(struct timerQueue));
new_timer->startTime = currTime;
new_timer->seqnum = nextSeqNum_A;
new_timer->next = NULL;
```

```
if(top == NULL){
  new_timer->next = top;
  top = new_timer;
  end = top;
}else{
  end->next = new_timer;
  end = new_timer;
}
```

- Timer will be running for the packet which is at the top of the queue.
- When timer interrupts, the first packet in the queue is re transmitted and this node gets removed from the queue.
- A new node of the re transmitted packet is created with the start time at which this packet was re transmitted and this node is added at rear end of the queue.
- Timer is again started for the first node in the queue for the remaining duration.

- Timeout remaining is calculated as below:

```
timeUtilized = get_sim_time() - itr->startTime;
float timeLeft = TIMEOUT - timeUtilized;
starttimer(0, timeLeft);
timerRunning_A = 1;
```

- **Therefore, timer runs in circle by continuously updating the queue whenever timer interrupts.**

Receiver Window

- In SR, unlike GBN, receiver window is maintained, and continuous correct packets are transmitted to layer5 which are within the window.

```
void A_timerinterrupt()
{
  starttimer(0, TIMEOUT);
  int i = base;
  while(i < nextSeqNum){
    new_packet = (struct pkt){0};
    new_packet.seqnum = i;
    strncpy(new_packet.payload, messageBuffer[i].data, 20);
    int checksum = compute_checksum(new_packet);
    new_packet.checksum = checksum;
    tolayer3(0, new_packet);
    i++;
  }
}
```

TEST RESULTS OF SR PROTOCOLS

SANITY

```
Running simulator [10 Runs] ...
Run#1 [seed=1234] ...  Done!
Run#2 [seed=1111] ...  Done!
Run#3 [seed=2222] ...  Done!
Run#4 [seed=3333] ...  Done!
Run#5 [seed=4444] ...  Done!
Run#6 [seed=5555] ...  Done!
Run#7 [seed=6666] ...  Done!
Run#8 [seed=7777] ...  Done!
Run#9 [seed=8888] ...  Done!
Run#10 [seed=9999] ...  Done!
PASS!
SANITY TESTS:  PASS
highgate {~/cse489589_assignment2/grader} > ./sanity_tests -p ../saqibhus/sr -r ./run_experiments
```

BASIC

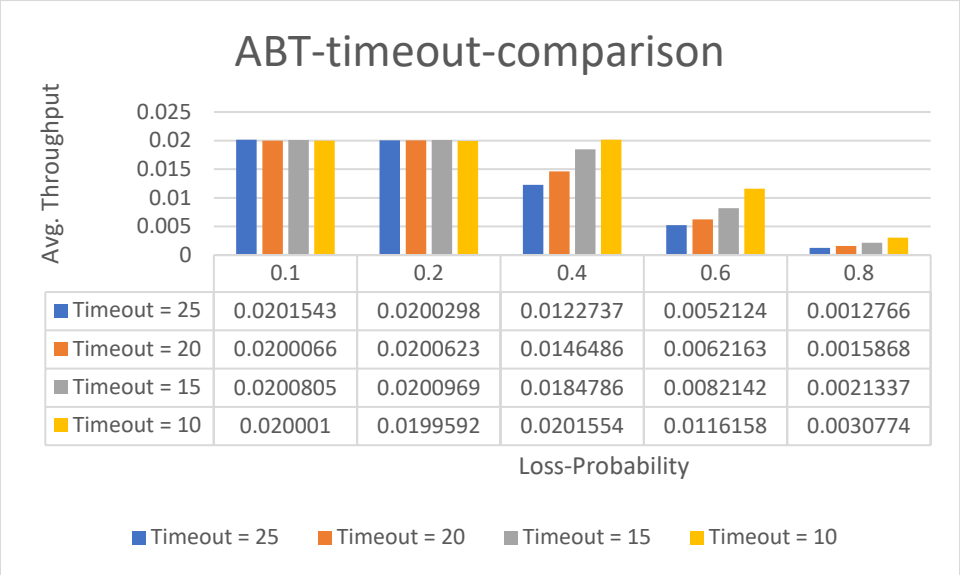
```
Running simulator [10 Runs] ...
Run#1 [seed=1234] ...  Done!
Run#2 [seed=1111] ...  Done!
Run#3 [seed=2222] ...  Done!
Run#4 [seed=3333] ...  Done!
Run#5 [seed=4444] ...  Done!
Run#6 [seed=5555] ...  Done!
Run#7 [seed=6666] ...  Done!
Run#8 [seed=7777] ...  Done!
Run#9 [seed=8888] ...  Done!
Run#10 [seed=9999] ...  Done!
PASS!
BASIC TESTS:  PASS
highgate {~/cse489589_assignment2/grader} > ./basic_tests -p ../saqibhus/sr -r ./run_experiments
```

ADVANCED

```
Running simulator [10 Runs] ...
Run#1 [seed=1234] ...  Done!
Run#2 [seed=1111] ...  Done!
Run#3 [seed=2222] ...  Done!
Run#4 [seed=3333] ...  Done!
Run#5 [seed=4444] ...  Done!
Run#6 [seed=5555] ...  Done!
Run#7 [seed=6666] ...  Done!
Run#8 [seed=7777] ...  Done!
Run#9 [seed=8888] ...  Done!
Run#10 [seed=9999] ...  Done!
PASS!
ADVANCED TESTS:  PASS
highgate {~/cse489589_assignment2/grader} > ./advanced_tests -p ../saqibhus/sr -r ./run_experiments
```

TIMEOUT SELECTION FOR DIFFERENT PROTOCOLS

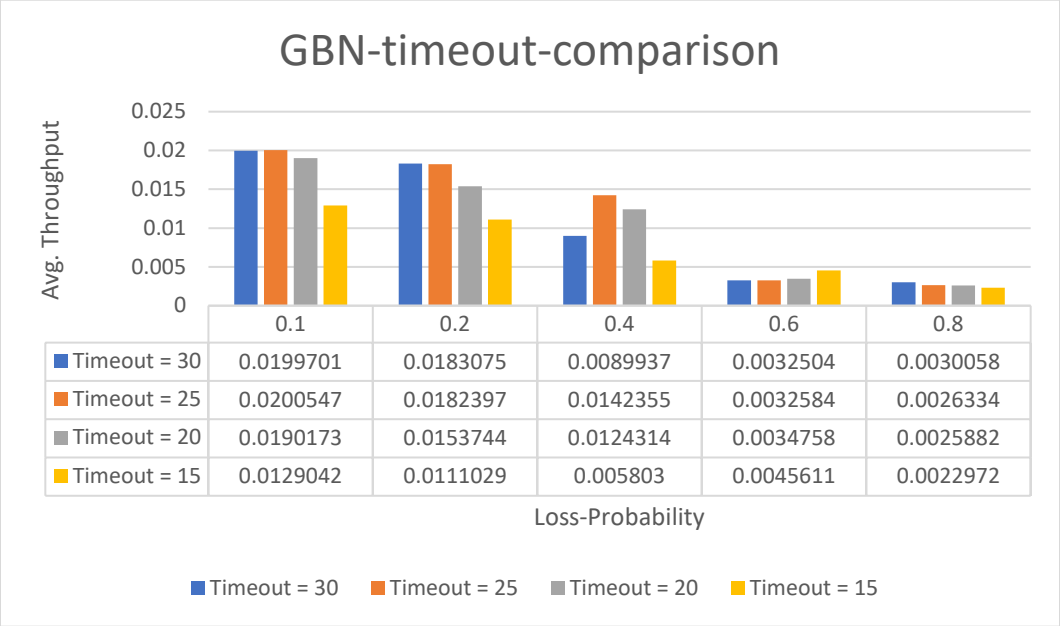
ABT Protocol



	0.1	0.2	0.4	0.6	0.8
Timeout = 25	0.0201543	0.0200298	0.0122737	0.0052124	0.0012766
Timeout = 20	0.0200066	0.0200623	0.0146486	0.0062163	0.0015868
Timeout = 15	0.0200805	0.0200969	0.0184786	0.0082142	0.0021337
Timeout = 10	0.020001	0.0199592	0.0201554	0.0116158	0.0030774

- As we can see from the above graph the yellow highlighted bar gives the best comparative throughput for different loss-probability.
- Hence, we have selected **timeout = 10** for ABT protocol.

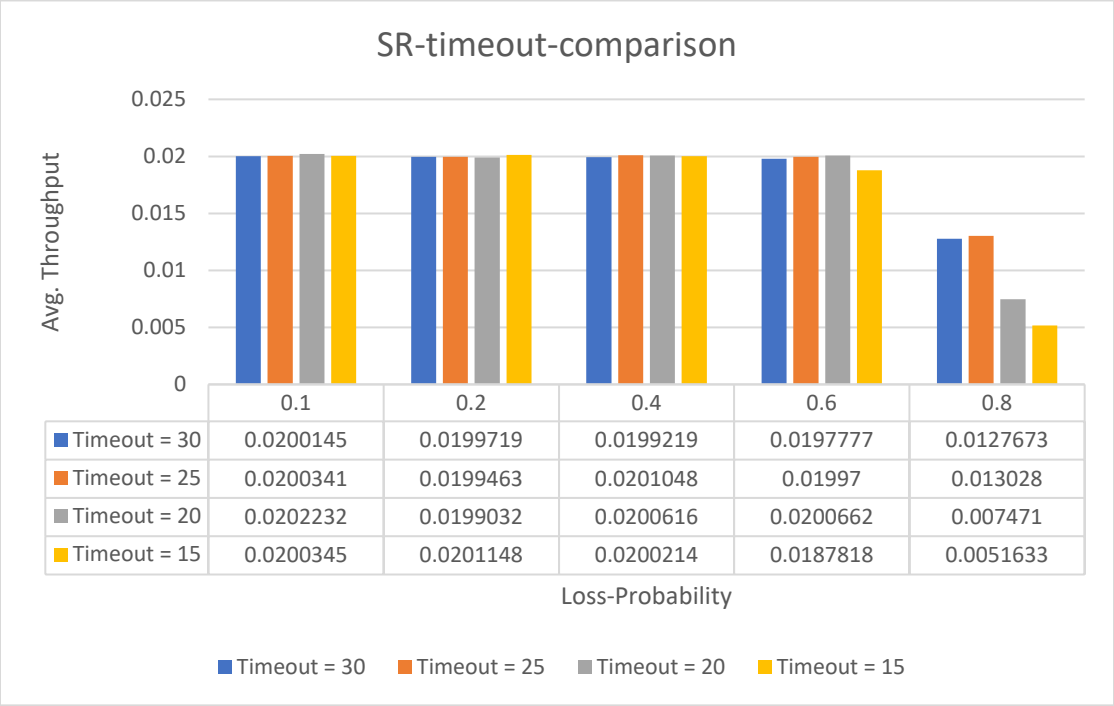
GBN Protocol



	0.1	0.2	0.4	0.6	0.8
Timeout = 30	0.0199701	0.0183075	0.0089937	0.0032504	0.0030058
Timeout = 25	0.0200547	0.0182397	0.0142355	0.0032584	0.0026334
Timeout = 20	0.0190173	0.0153744	0.0124314	0.0034758	0.0025882
Timeout = 15	0.0129042	0.0111029	0.005803	0.0045611	0.0022972

- As we can see from the above graph the orange highlighted bar gives the best comparative throughput for different loss-probability.
- Hence, we have selected **timeout = 25** for GBN protocol.

SR Protocol



	0.1	0.2	0.4	0.6	0.8
Timeout = 30	0.0200145	0.0199719	0.0199219	0.0197777	0.0127673
Timeout = 25	0.0200341	0.0199463	0.0201048	0.01997	0.013028
Timeout = 20	0.0202232	0.0199032	0.0200616	0.0200662	0.007471
Timeout = 15	0.0200345	0.0201148	0.0200214	0.0187818	0.0051633

- As we can see from the above graph the orange highlighted bar gives the best comparative throughput for different loss-probability.
- Hence, we have selected **timeout = 25** for SR protocol.

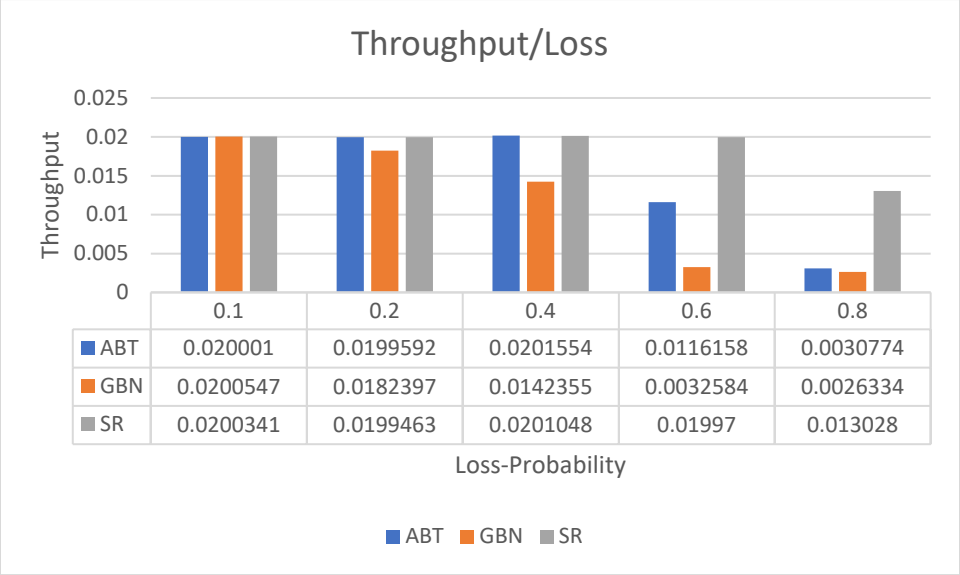
Experiment 1

CUMULATIVE GRAPHS (ABT, GBT & SR)

Window size = 10

	0.1	0.2	0.4	0.6	0.8
ABT	0.020001	0.0199592	0.0201554	0.0116158	0.0030774
GBN	0.0200547	0.0199428	0.0174943	0.0019141	0.0004852
SR	0.0200341	0.0199463	0.0200581	0.0180341	0.0043721

Window size = 50



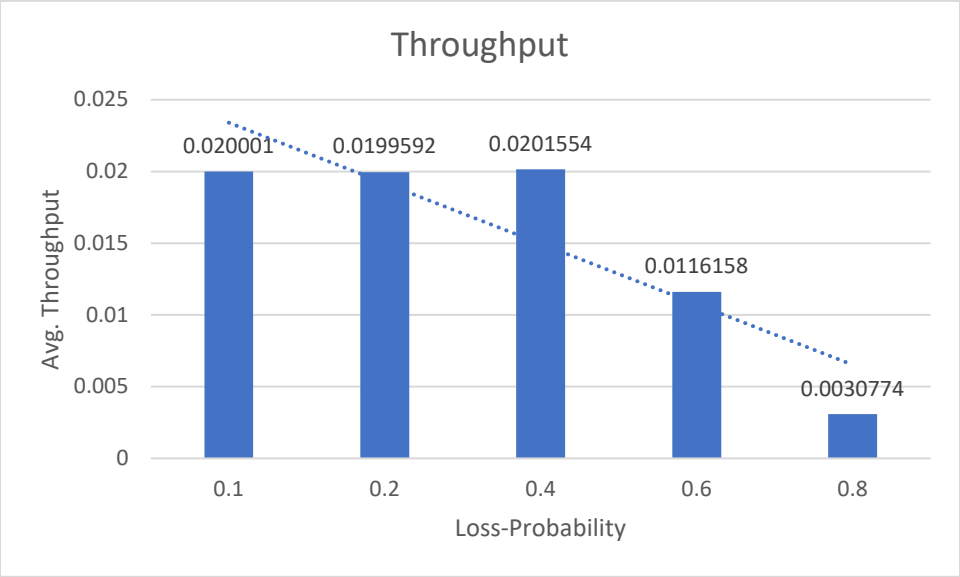
	0.1	0.2	0.4	0.6	0.8
ABT	0.020001	0.0199592	0.0201554	0.0116158	0.0030774
GBN	0.0200547	0.0182397	0.0142355	0.0032584	0.0026334
SR	0.0200341	0.0199463	0.0201048	0.01997	0.013028

- The SR protocol performs better than the other two protocols.

- Performance of the SR protocol is significantly improved for large window size, and it performance gap widens when the loss is huge as well as the window size is large.
- The GBN protocol performs worst when the loss is high.
- ABT protocol works well when there is less loss.

INDIVIDUAL GRAPHS

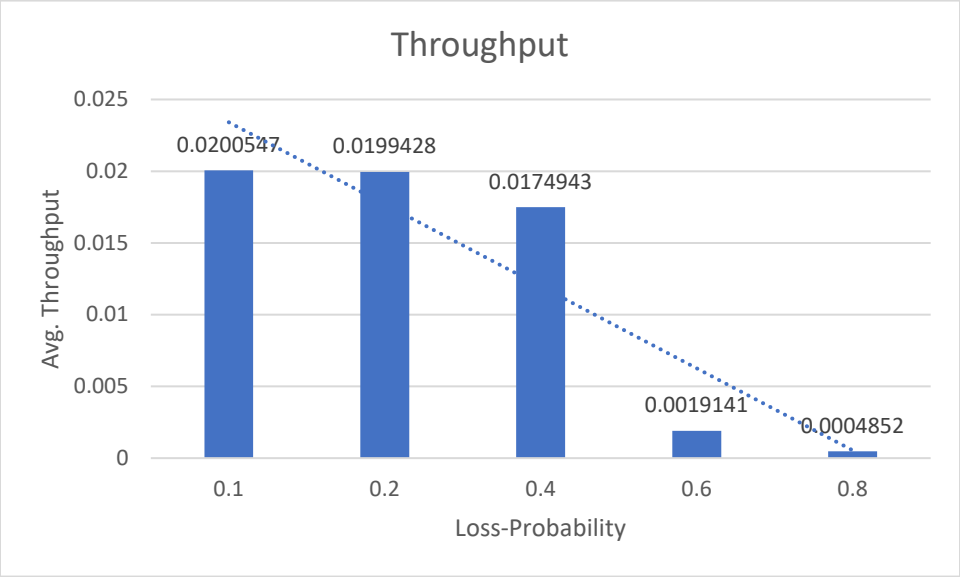
ABT Protocol



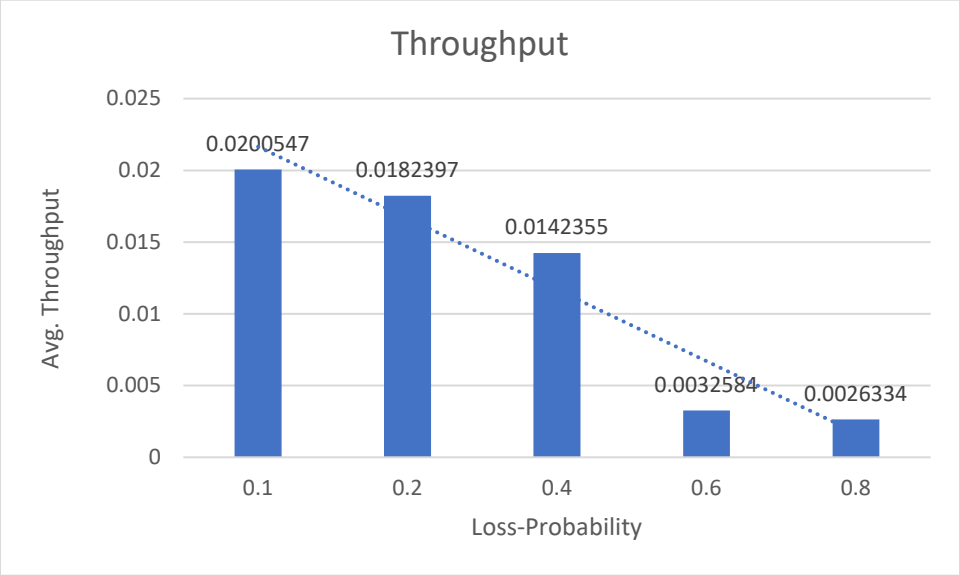
- The ABT protocol doesn't have the concept of window.
- This protocol performs well for less loss. But performance drops sharply when there is more loss.

GBN protocol

Window size = 10



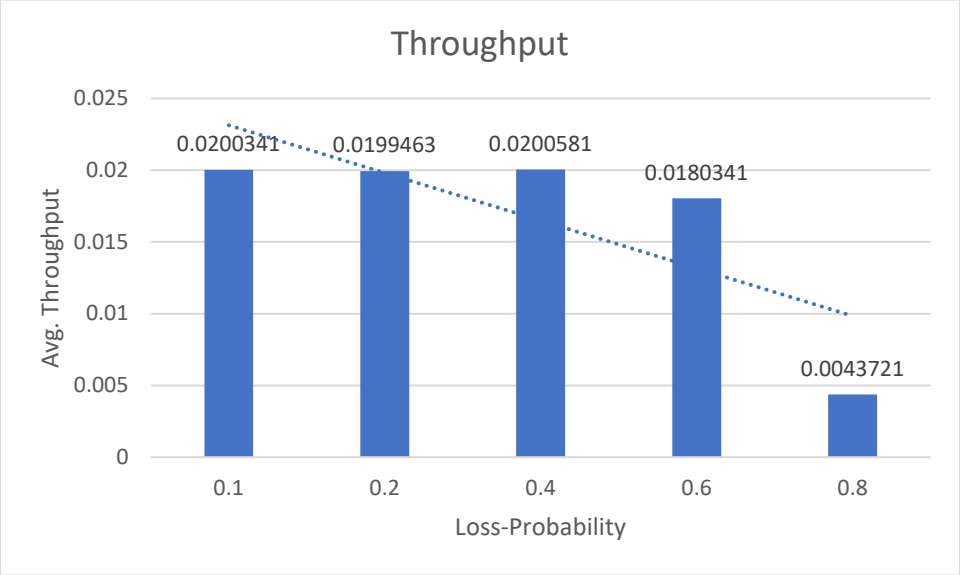
Window size = 50



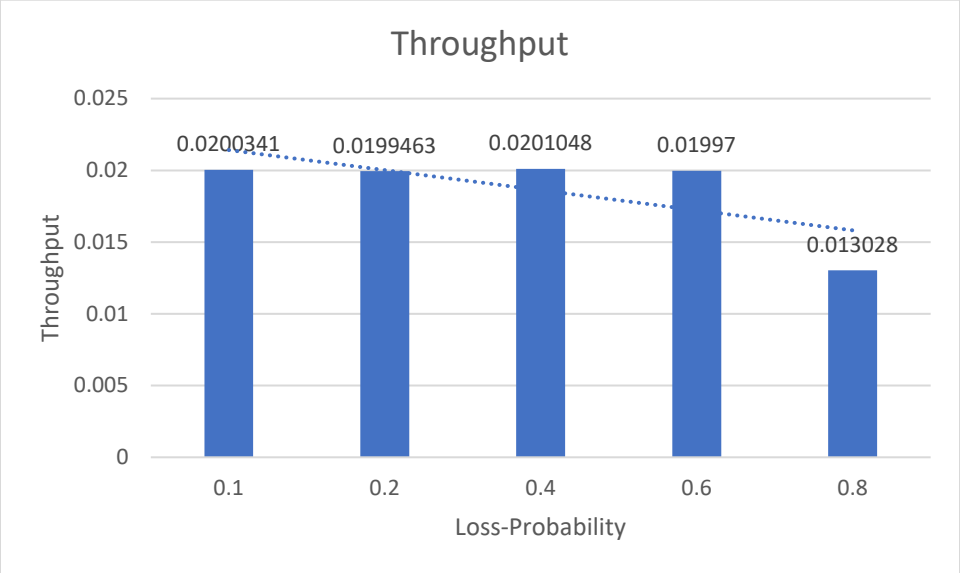
- We can observe from the following graphs that GBN protocol works better when the window size is large i.e., 50 in this case.
- Performance of GBN drops drastically when we increase the loss.

SR protocol

Window size = 10



Window size = 50

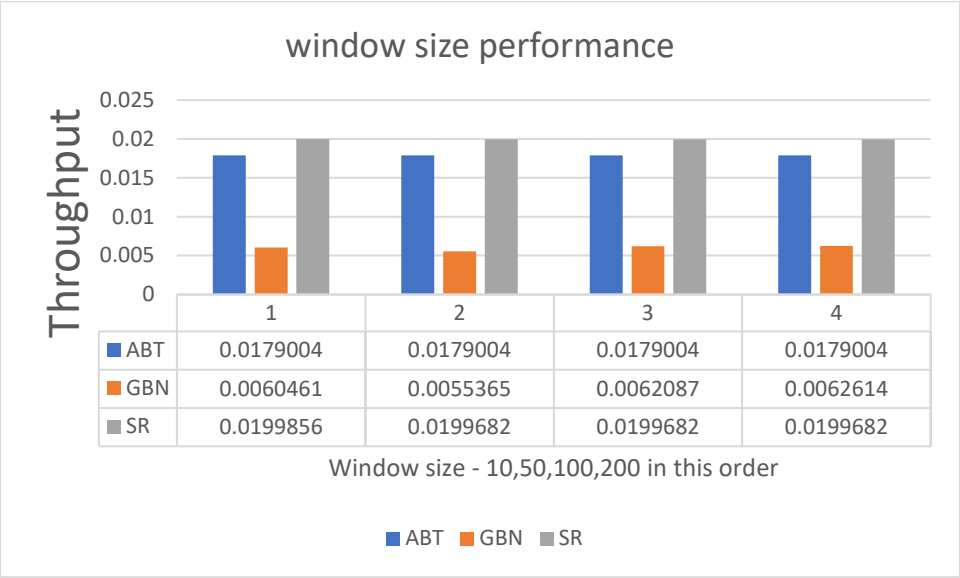


- We can observe from the following graphs that SR protocol works better when the window size is large i.e., 50 in this case.
- We are getting significantly high throughput when we have large window.
- It performs significantly better than GBN under higher loss-probability.

EXPERIMENT 2

For loss probability 0.2 -

For loss probability 0.5 –



For loss probability 0.8 –