**CSE 589**
**Programming Assignment 2**
**Reliable Data Transfer Protocol**

# Analysis Report

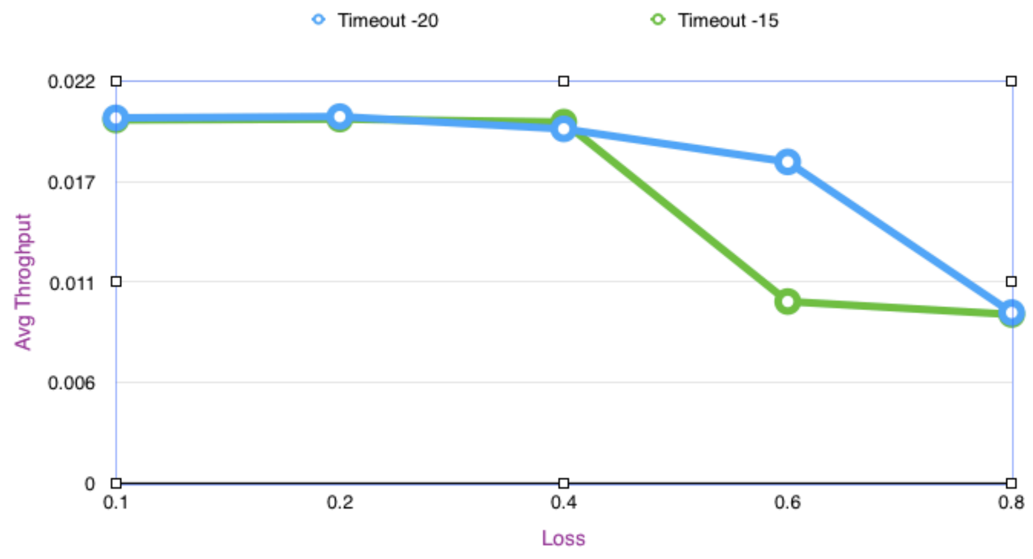
**I have read and understood the course academic integrity policy.**

**Shivam Sahu**
**#50247673**

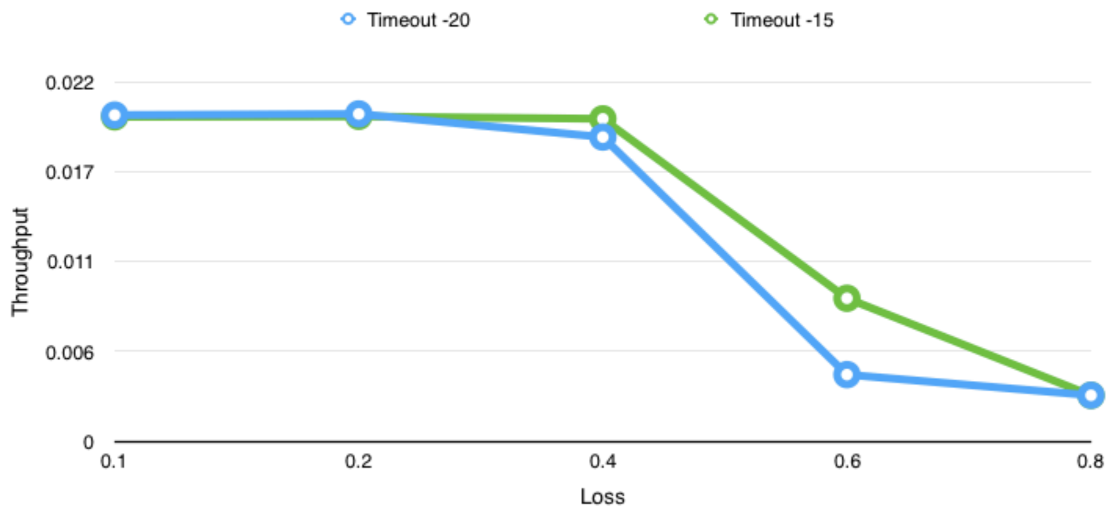
## 1. Timeout scheme :

a) **ABT** - This protocol used a constant timeout of 20 time unit. I used different timeout with different loss probabilities and observed that 20 unit timeout delivers the best result among all the timeout.

b) **GBN** - For this protocol, I used different timeout with different loss probabilities for different window size. At the end , I observed the most optimal timeout for GBN is 20 time unit. These are some graph which I used for observation:-

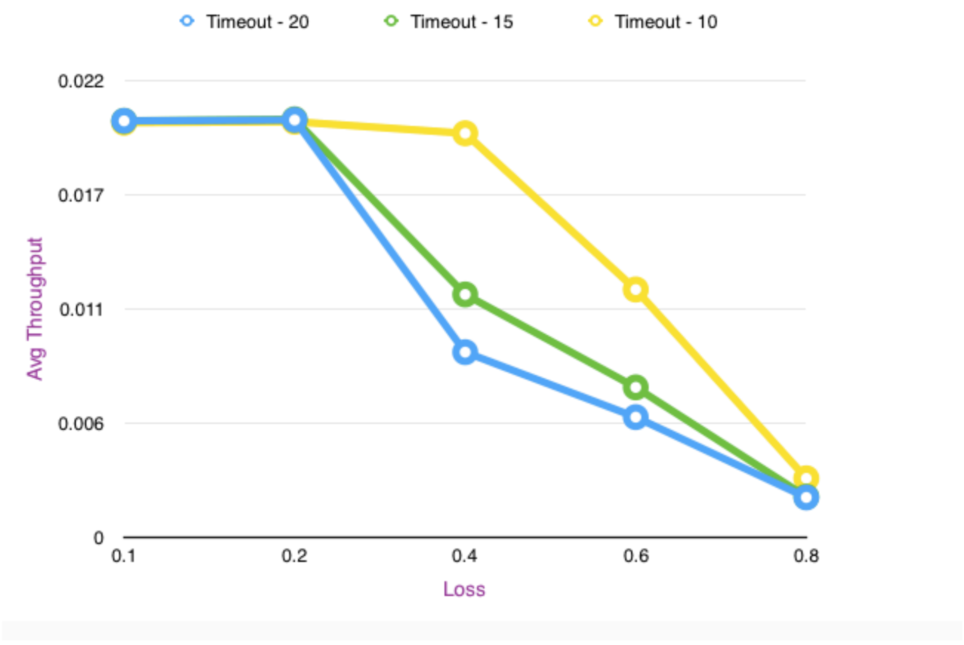


***This graph is for window size - 10***

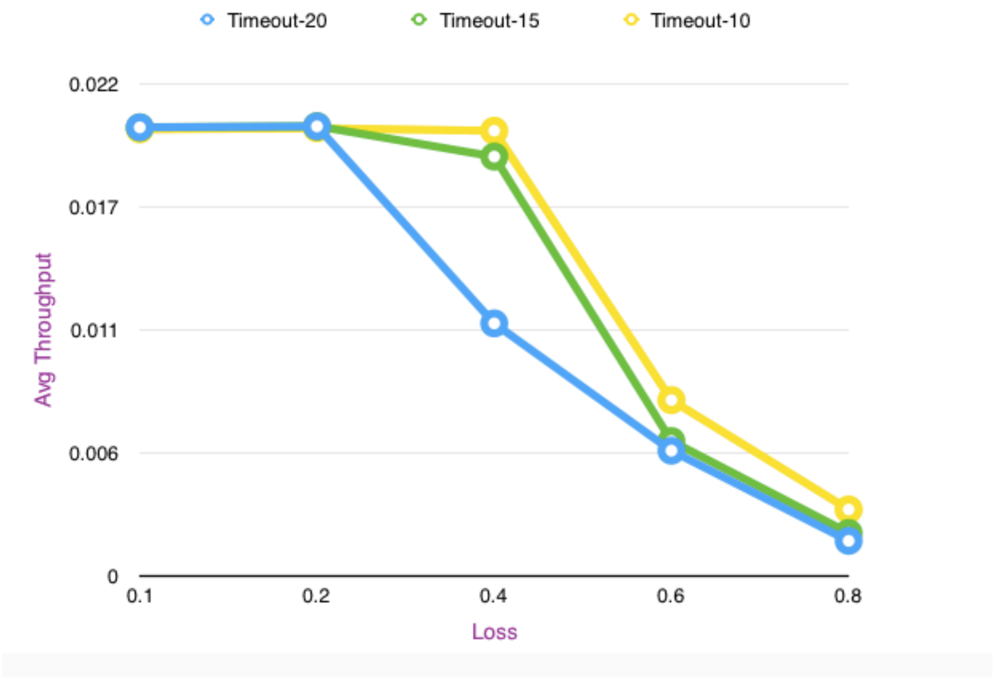


***This graph is for window size - 50***

c) **SR** - For this protocol, I also used different timeout with different loss probabilities for different window size. At the end , I observed  the most optimal timeout for SR is 10 time unit. These are some graph which I used for observation:-



***This graph is for window size - 10***



***This graph is for window size - 50***

**2. Multiple software timers in SR :**

I have used a vector of type struct A_pktData. A_pkdata contain three important information.1st is a packet. 2nd is timestamp when the packet is sent. 3rd is a flag which defines that packet is acknowledged or not.
This my struct A_pktData -

```
struct A_pktData{
  struct pkt *packet;
  int startTime;
  bool acknowledged;
  A_pktData(struct pkt *pack)
  {
    packet = pack;
    int startTime = 0;
    bool acknowledged = false;
  }
};
```

So whenever a timeout happens, I find a packet which has minimum timestamp inside sender window and resends it and also set the new timestamp for it. Then, again I find a packet which has minimum timestamp inside sender window and set my timeout according to this packet.

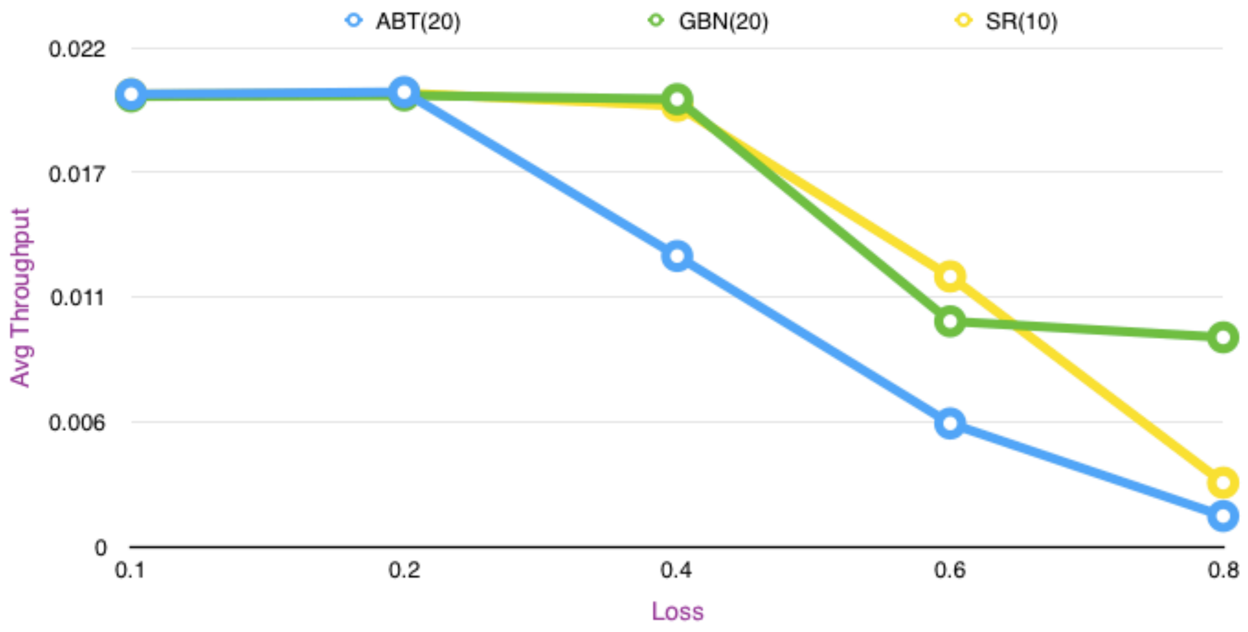
In other words, I am finding two packets which have the lowest timestamp inside sender window. Then I am resending the lowest timestamp packet and setting the timeout according to the second lowest timestamp packet.

## EXPERMIENT 1

### a) Window Size - 10

Experiment 1( Window Size - 10)

| Loss | ABT(20) | GBN(20) | SR(10) |
|---|---|---|---|
| 0.1 | 0.0199543 | 0.0198522 | 0.0199508 |
| 0.2 | 0.0200457 | 0.0198991 | 0.0199966 |
| 0.4 | 0.0128183 | 0.0197323 | 0.0194395 |
| 0.6 | 0.005439 | 0.0099412 | 0.0119185 |
| 0.8 | 0.0013576 | 0.0092345 | 0.0028271 |

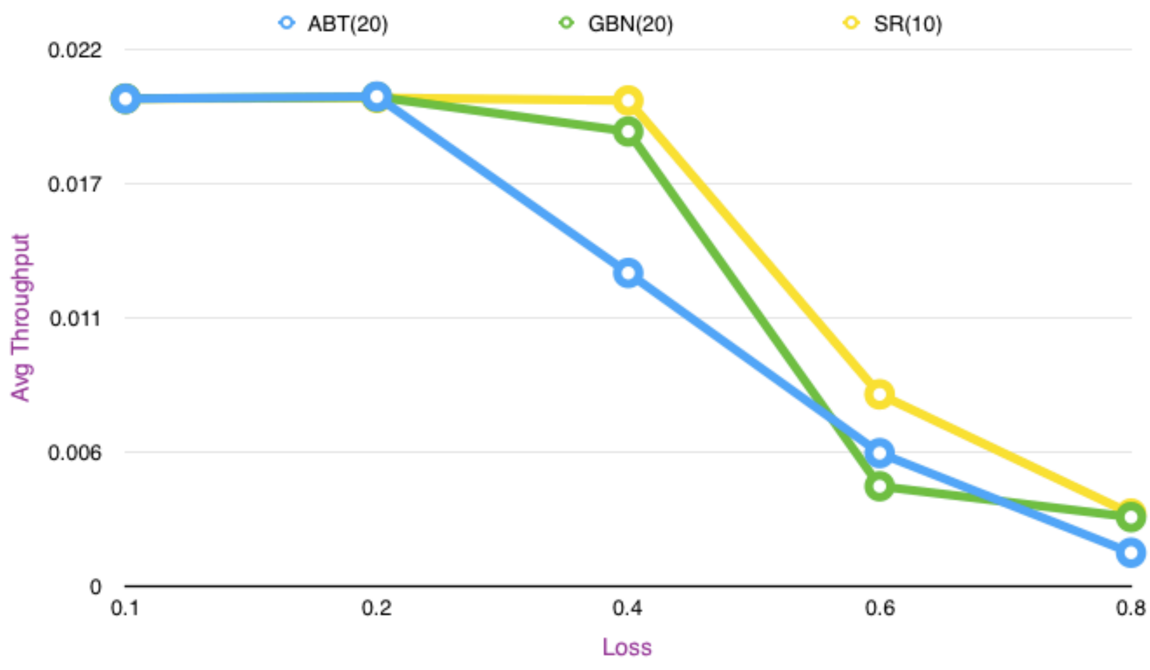


**As we can see from the graph that when we increase the loss probability our throughput will decrease. Up to certain loss probability, all three protocol generate the same throughput. After loss probability 0.4, SR and GBN give comparable throughput and ABT gives the lowest throughput. At 0.8 loss probability, GBN is perming good than SR. It concludes that for lower window size GBN is performing well than SR because, on the timer interrupt, GBN send its whole window, which is very small in this scenario.**

## b) Window Size - 50

Experiment 1( Window Size - 50)

| Loss | ABT(20) | GBN(20) | SR(10) |
|---|---|---|---|
| 0.1 | 0.0199543 | 0.0199625 | 0.0199508 |
| 0.2 | 0.0200457 | 0.0200313 | 0.0199966 |
| 0.4 | 0.0128183 | 0.0186136 | 0.0198837 |
| 0.6 | 0.005439 | 0.0040806 | 0.0078468 |
| 0.8 | 0.0013576 | 0.0028173 | 0.0029571 |

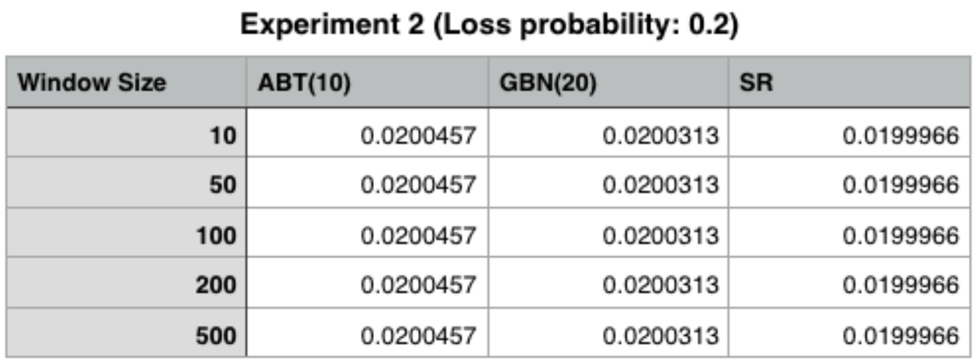


**As we can see from the graph, when we increase the loss probability our throughput will decrease. Up to loss probability 0.2, all three protocol generate same throughput. After loss probability 0.2, SR gives highest throughput then GBN and ABT give lowest throughput.**

**It concludes SR is the winner here. GBN is performing badly because of the overhead of retransmission of large number of packets due to the larger window size. ABT doesn’t depend on the window size.**

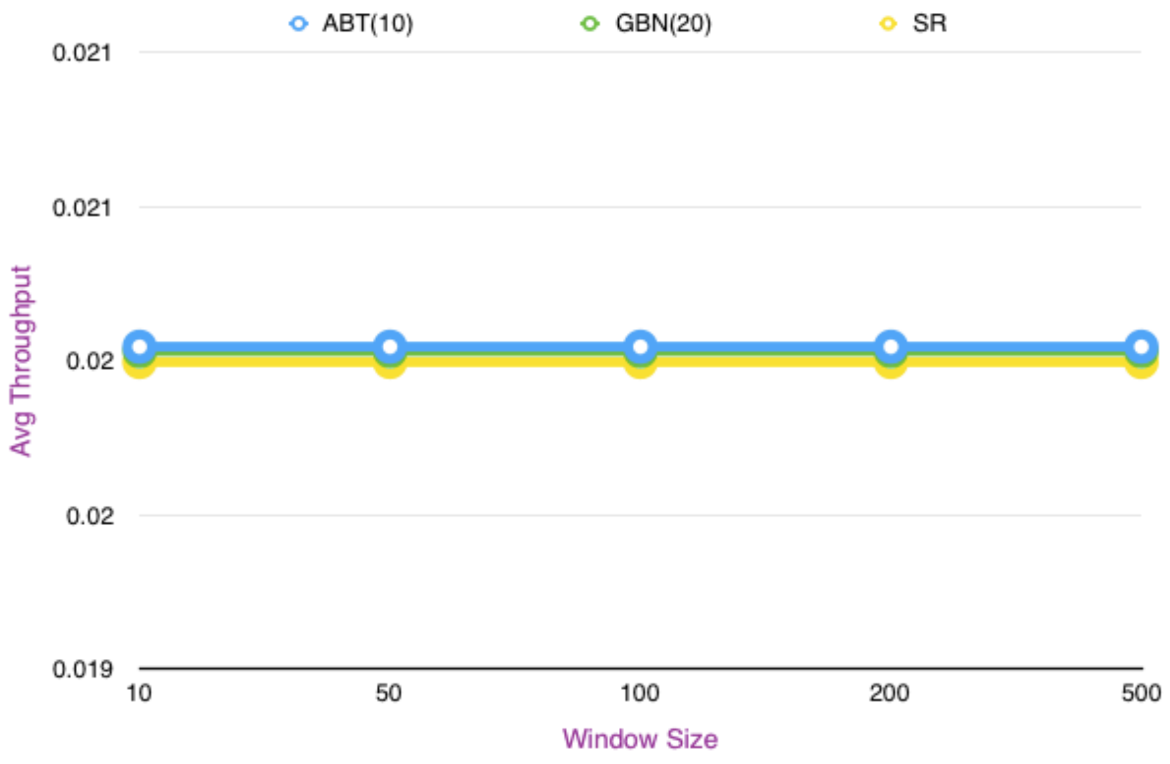
## EXPERMIENT 2

### a) Loss probability - 0.2



**Experiment 2 (Loss probability: 0.2)**

| Window Size | ABT(10) | GBN(20) | SR |
|---|---|---|---|
| 10 | 0.0200457 | 0.0200313 | 0.0199966 |
| 50 | 0.0200457 | 0.0200313 | 0.0199966 |
| 100 | 0.0200457 | 0.0200313 | 0.0199966 |
| 200 | 0.0200457 | 0.0200313 | 0.0199966 |
| 500 | 0.0200457 | 0.0200313 | 0.0199966 |

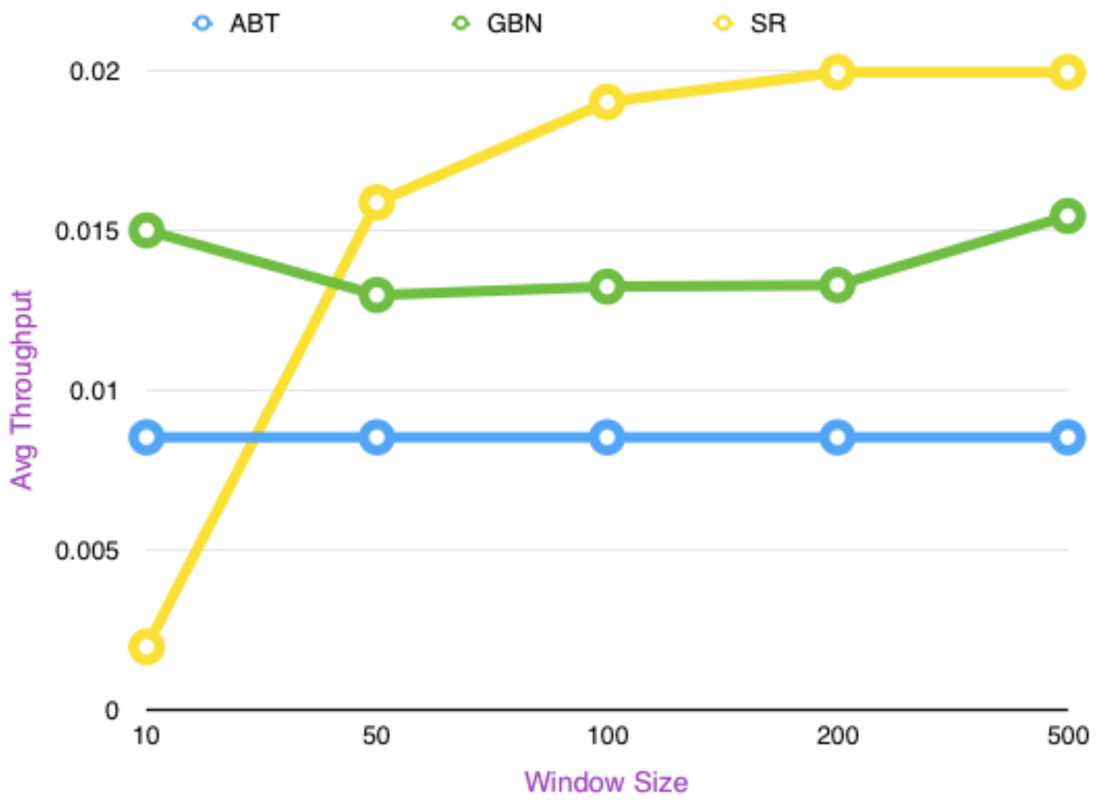


**In this, all the three protocol gives same (comparably) throughput on all window size. It’s because of loss probability 0.2 which is very low, so every protocol performs very well with respect to any window size.**

**b) Loss probability - 0.5**

**Experiment 2 (Loss probability: 0.5)**

| Windowsize | ABT | GBN | SR |
|---|---|---|---|
| 10 | 0.0085084 | 0.0149813 | 0.001955 |
| 50 | 0.0085084 | 0.0129581 | 0.0158547 |
| 100 | 0.0085084 | 0.0132213 | 0.0189955 |
| 200 | 0.0085084 | 0.0132669 | 0.0199276 |
| 500 | 0.0085084 | 0.0154308 | 0.0199276 |

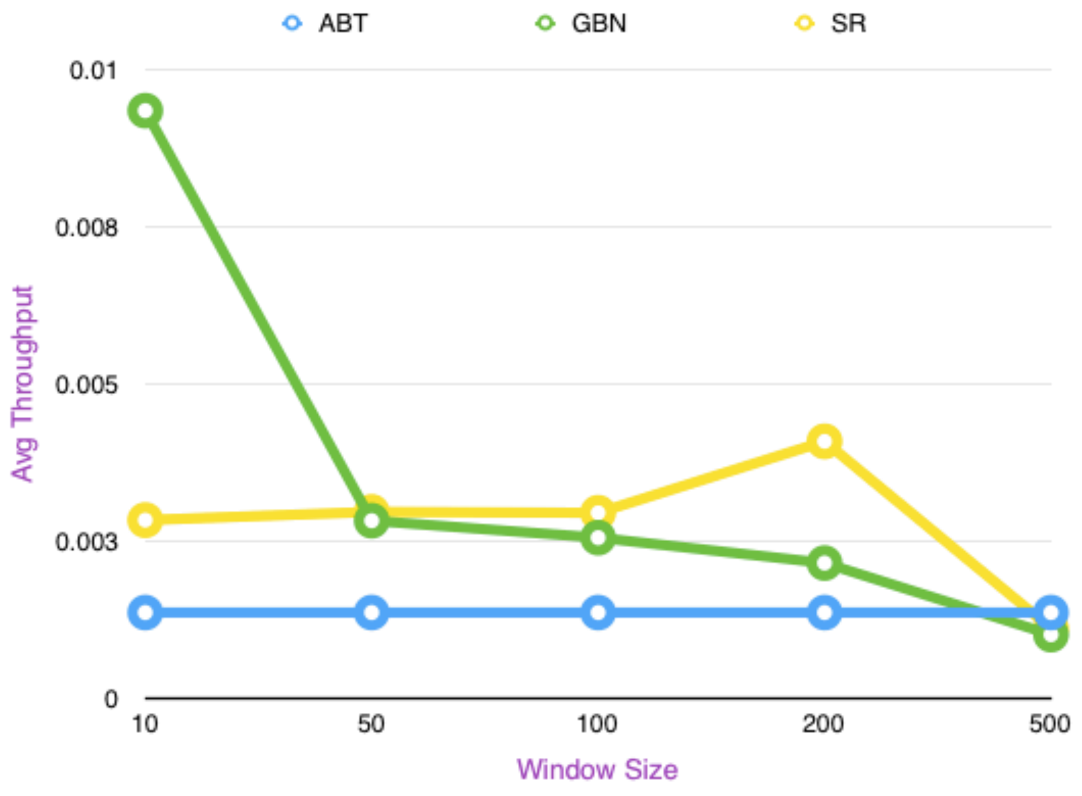


**In this case, at window size 10, GBN perform well than SR but as window size increase SR is performing well then GBN. At higher window size, GBN retransmits a large number of packets which will increase the overhead of protocol. ABT is independent of window size.**

**c) Loss probability - 0.8**

**Experiment 2 (Loss probability: 0.8)**

| Window Size | ABT | GBN | SR |
|---|---|---|---|
| 10 | 0.0013576 | 0.0093367 | 0.0028271 |
| 50 | 0.0013576 | 0.0028173 | 0.0029571 |
| 100 | 0.0013576 | 0.0025479 | 0.0029435 |
| 200 | 0.0013576 | 0.0021444 | 0.0040793 |
| 500 | 0.0013576 | 0.0010061 | 0.0011061 |



**In this case, at window size 10, GBN perform well than SR but as window size increase SR is performing well then GBN. At higher window size, GBN retransmits a large number of packets which will increase the overhead of protocol. ABT is independent of window size so its throughput is constant.**