# Reliable Transport Protocols **I have read and understood the course academic integrity policy.** Shefali Sharma, 50247677

**Introduction**

In this programming assignment, we implemented the sending and receiving transport-layer code for implementing a simple reliable data transfer protocol. There are 3 versions of this assignment, the Alternating-Bit Protocol version, the Go-Back-N version, and the Selective-Repeat version.

The overall structure of the environment is shown below:

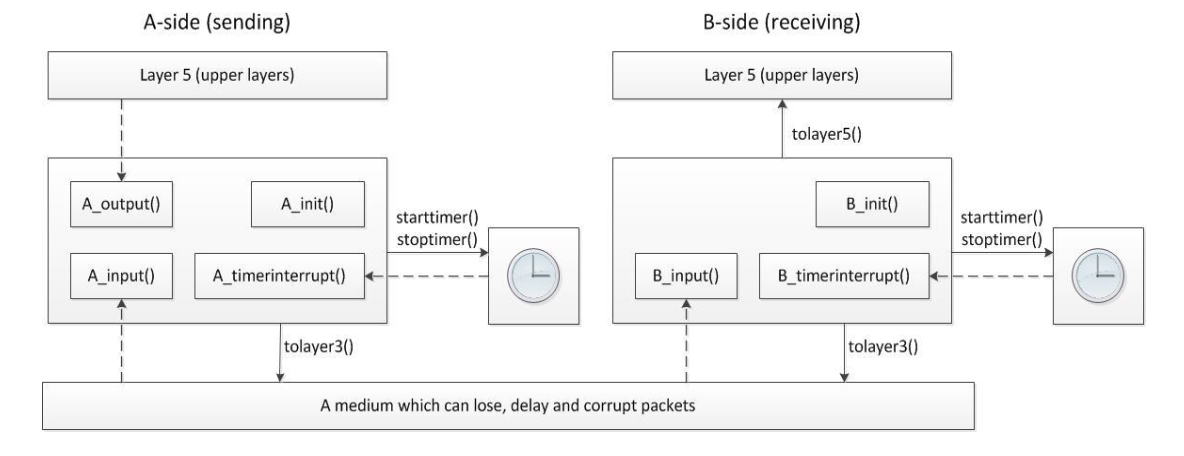


Fig: 1

**Implementation of Different Transport Protocols**

**Alternating-Bit Protocol**  
Implementation:

* Messages are received by the application layer at ‘A\_output()’ (Sender)
* All the messages received by the Sender from the Application Layer are stored in a buffer.
* Sender will send the message for the sequence number and then wait for the acknowledgement of the sent packet. Sender will also start the timer for the sent packet.
* Receiver (B\_input()) will first check if the received packet has the sequence number as that of the “expected sequence”, and also check if the packet is corrupted.
* If a valid packet (with correct sequence number and un-corrupted data) is received, receiver forwards the message to Layer5 and sends an acknowledgement back to the Sender (A\_input()). Sender also updates the “expected sequence” number.
* If the packet received by the Receiver (B\_input())is an un-corrupted packet but the sequence number is not as expected, then the receiver only sends the acknowledgement back to the Sender (This is done so that loss/corruption of ACK from Receiver to Sender can be taken care of).
* Once the Sender receives the acknowledgement, it then checks if a valid (for the correct sequence number and uncorrupted) acknowledgement is received.
* If the acknowledgement received by the Sender is valid, it then stops the timer and then sends the next message from the buffer (if the buffer is not empty) with updated sequence number and resets the timer again.
* In case of a loss/corruption of a packet from the Sender to the receiver or a loss/corruption of the acknowledgement to the Sender by the receiver, the Sender waits for the “time-interrupt” to occur, upon which it resends the lost or the corrupted packet to the Receiver.

Timeout Scheme Used:

After closely monitoring the protocol under different combinations of timeouts and loss probabilities, I have used a fixed timeout value of 20.

**Go-Back-N Protocol**

Implementation:

* Messages are received by the application layer at ‘A\_output()’ (Sender)
* All the messages received by the Sender from the Application Layer are stored in a buffer.
* Sender will first check if the “sequence number” of the message received from the Application Layer is more than or equal to the base value (which is initially assigned the value of the sequence number of the first packet created), and if the “sequence number” is within the window size assigned.
* If the “sequence number” of the packet satisfies the above mentioned criteria, it then sent to the Receiver (B\_input()), and the timer for the sent packet is started.
* Receiver first checks if the packet received has the sequence number as expected by the receiver and that the packet is not corrupted.
* If the packet is not corrupted and the “sequence number” is less than the expected, then it is assumed that the packet has already been received and an acknowledgement for the packet is again sent to the Sender.
* If the packet in not corrupted and the value of the packet’s “sequence number” is as expected, then the Receiver (B\_input()) creates an “acknowledgement packet”, and send the packet to the Sender with “acknum” equal to the “sequence number” of the packet received.
* Packet is then delivered to the Application Layer by the Receiver (B\_input()) and “expected sequence number” is incremented.
* After receiving the acknowledgement from the Receiver, the Sender (A\_input()) first checks if the acknowledgement received is not corrupt.
* If the acknowledgement received is not corrupted and “acknum” is equal to the base, then the base is incremented by one and all the messages in the buffer which lie within the window size are then again sent to the Receiver.
* If the “acknum” received by the Sender is more than the base, then the base is set to “acknum + 1”, as it is assumed that the acknowledgement of the packets for sequence number before “acknum” and after base could be lost, but “acknum” confirms delivery of correct packets at the receiver. Sender then sends all the packets from the updated base to (base + window size) to the Receiver.
* Sender also restarts the timer for the first packet that is sent by “A\_input()”.
* If a time-interrupt is encountered, then the Sender simply re-transmits all the packets starting from the base to (base + window size) to the Receiver.

Timeout Scheme Used:

After closely monitoring the protocol under different combinations of timeouts and loss probabilities, I have used a fixed timeout value of 40.

**Selective-Repeat Protocol**

Implementation:

* Messages are received by the application layer at ‘A\_output()’ (Sender)
* All the messages received by the Sender from the Application Layer are stored in a buffer.
* Sender will first check if the “sequence number” of the message received from the Application Layer is more than or equal to the base value (which is initially assigned the value of the sequence number of the first packet created), and if the “sequence number” is within the window size assigned.
* If the “sequence number” of the packet satisfies the above mentioned criteria, it then sent to the Receiver (B\_input()), and the timer for the sent packet is started.
* Receiver first checks if the received has the sequence number within the expected window-size range and that the packet is not corrupted.
* If the packet is not corrupted and the “sequence number” is within the expected window-size range, it is then stored in the “Receive buffer”, maintained by the Receiver (B\_input()).
* If the packet is already present in the “Receiver buffer”, then an acknowledgement for the packet is sent.
* If the packet is not present in the “Receive buffer”, then an acknowledgement is first sent to Sender, and the packet is then stored in the “Receive buffer”.
* If the “sequence number” of the packet is same as that of the base of the “Receive Buffer”, then the packet is delivered to the Application Layer by the Receiver, and all the subsequent packets stored in the receive buffer are also delivered. Here we update the base of the receive buffer by incrementing it whenever the packet is delivered to the Application Layer.
* Once the packet is received by the Sender, it then checks if the acknowledgement received is valid (within the window-size range, and is not corrupted).
* If the acknowledgement packet received is valid, then if the acknowledgement was not already received, then the packet is marked as acknowledged.
* If the packet received has the “acknum” same as that of the base, then the base is incremented to the next packet in the “Sender buffer”, which was not acknowledged.
* Timer is then reset according to the last sent packet by the receiver and if there is a new packet which is now available under the “window size”, it is sent from the “Sender buffer” to the Receiver.
* If time-interrupt occurs, then if the base is already marked as acknowledged, it is incremented to the next un-acknowledged packet.
* All the un-acknowledged packets starting from base to )base + window size) are then re-sent to the Receiver.

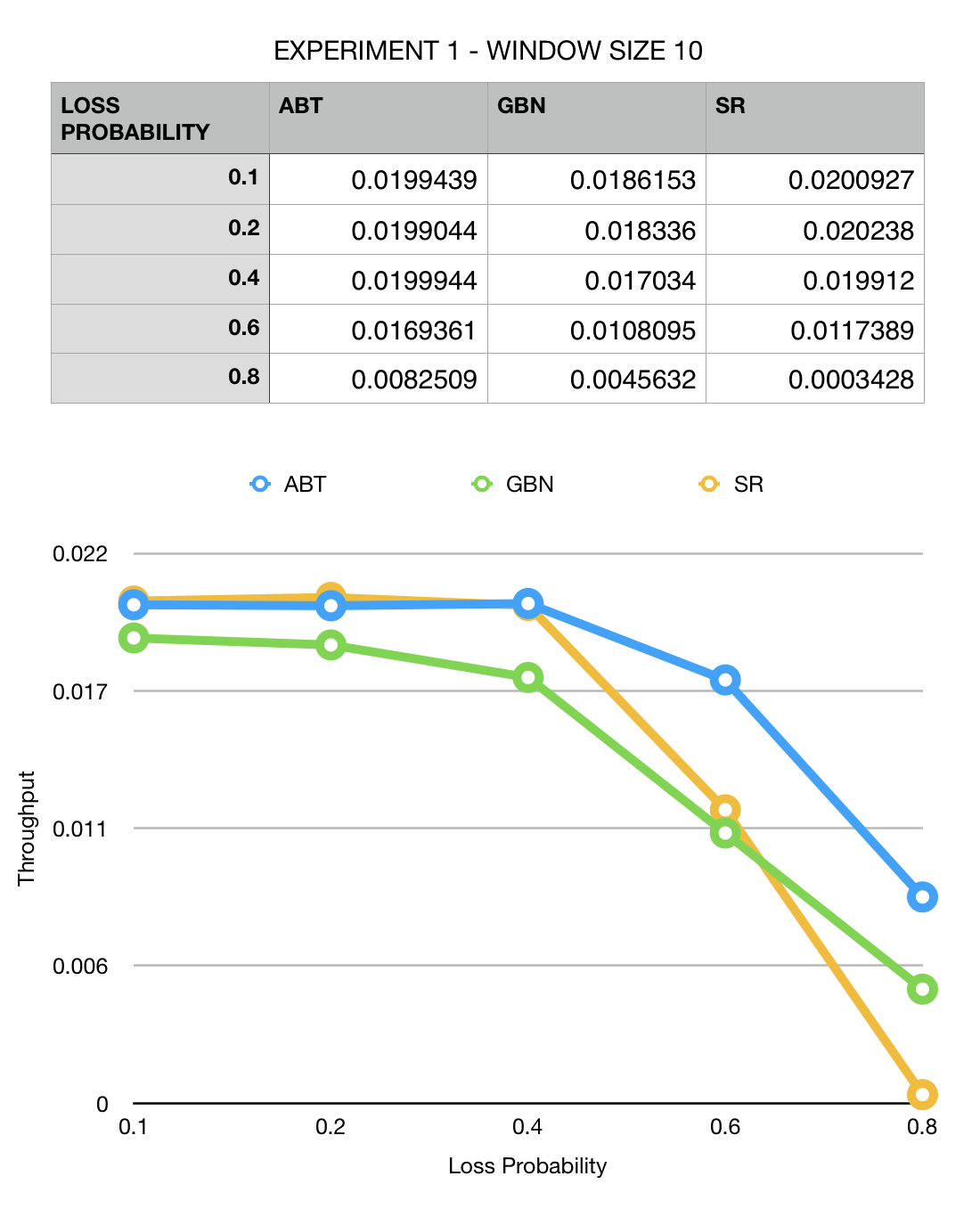
Timeout Scheme Used:

For SR, I used a slightly different implementation. For each packet, the time at which the packet is sent is stored.

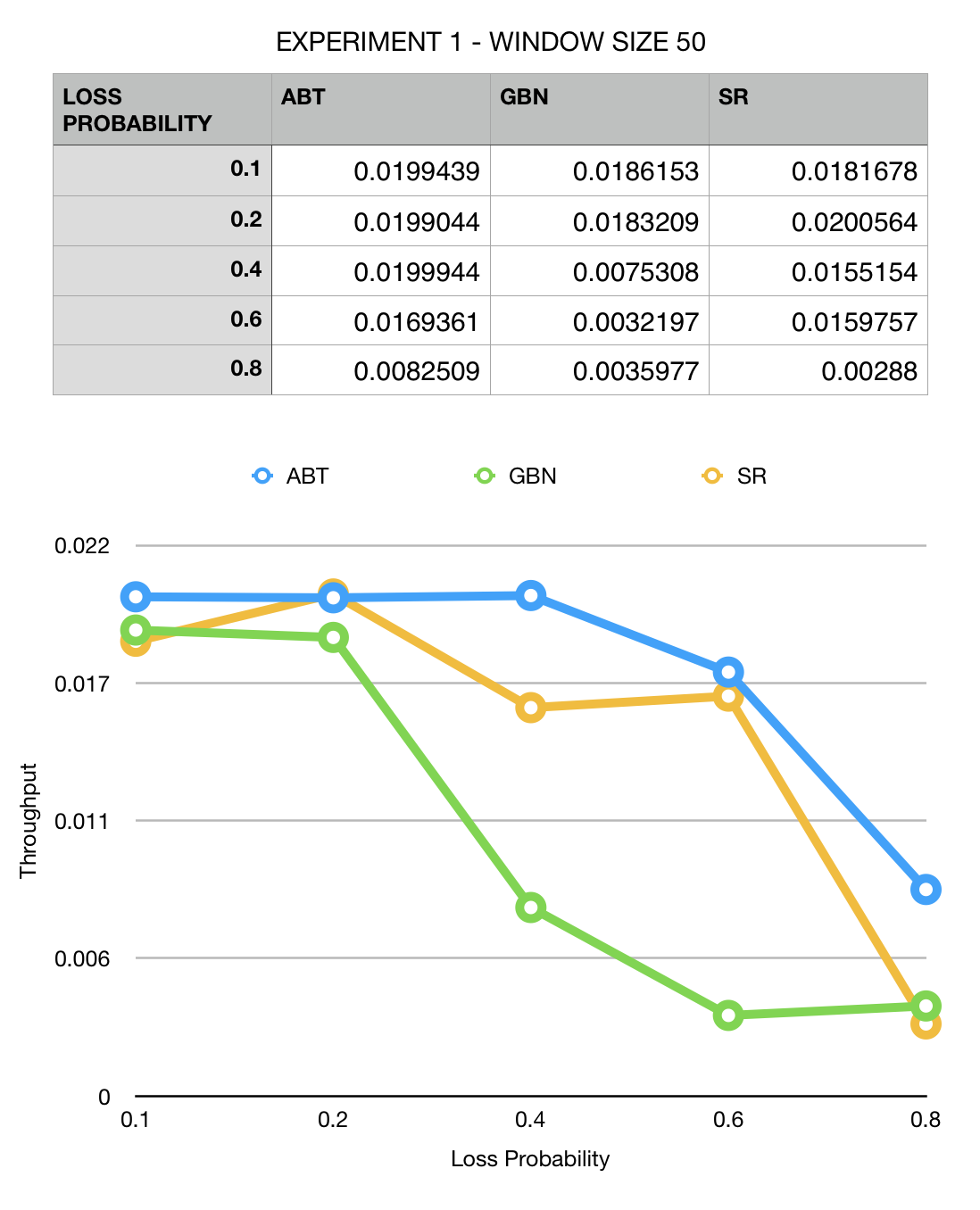
* If the acknowledgement packet received at the Sender (A\_input()) is such that the “acknum” is base and all the other packets till “base + window size” have already been acknowledged, then the base is updated and the timer is started by keeping the fixed time of 15.
* If the base is received but the all the other packets till “base + window size” are not received, then the timer’s time is set by calculating the delay as -   
  Delay = RTT - (Time of the simulation now – starttime of the first un-acknowledged packet), where RTT = 15.

**Performance Analysis Results:**

Experiment 1:



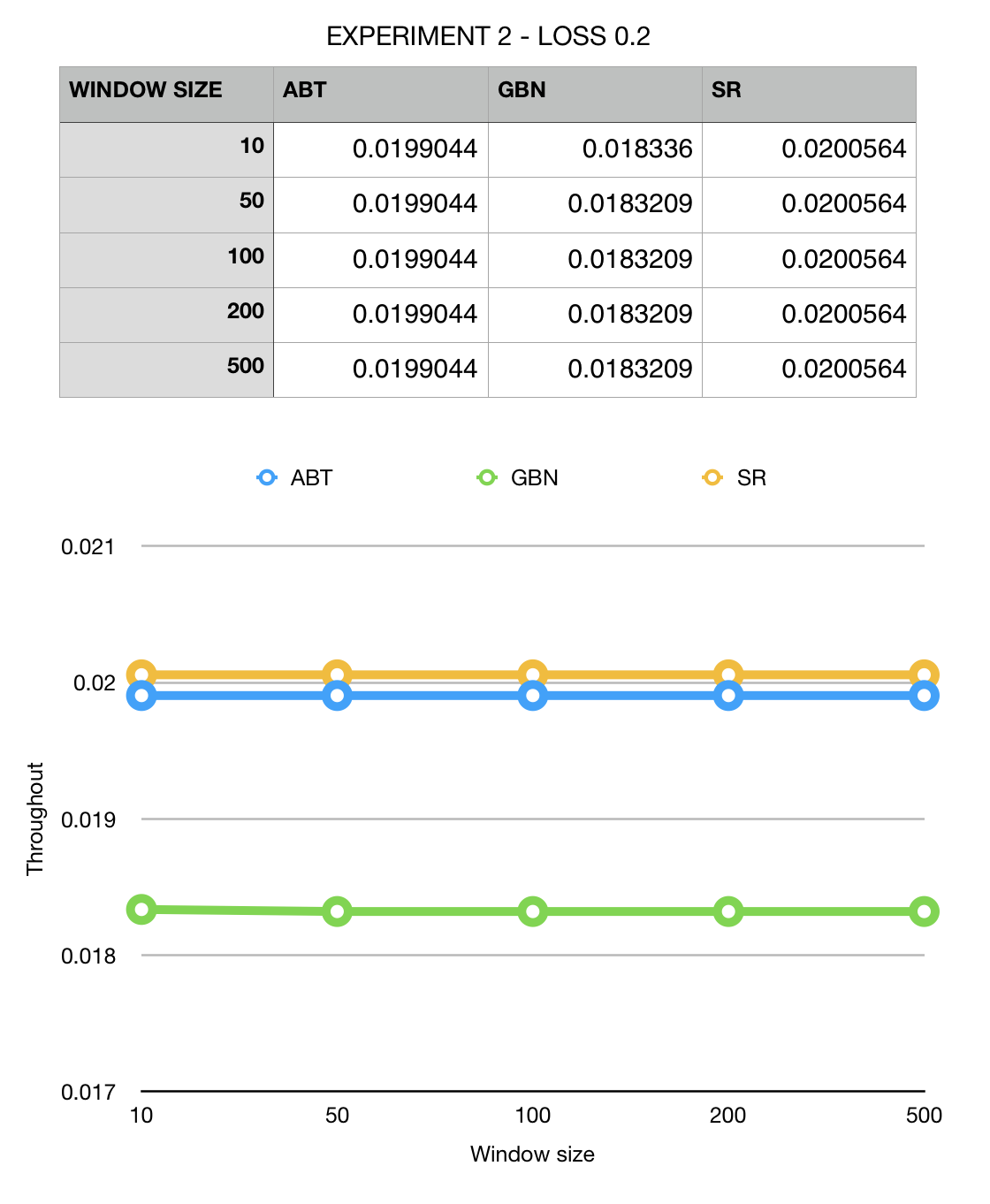
**Findings:**

* All the three protocols have almost the same topology.
* For loss 0.8, SR drops steeply.
* At certain points SR performs better than ABT, which is an expected output.

**Findings:**

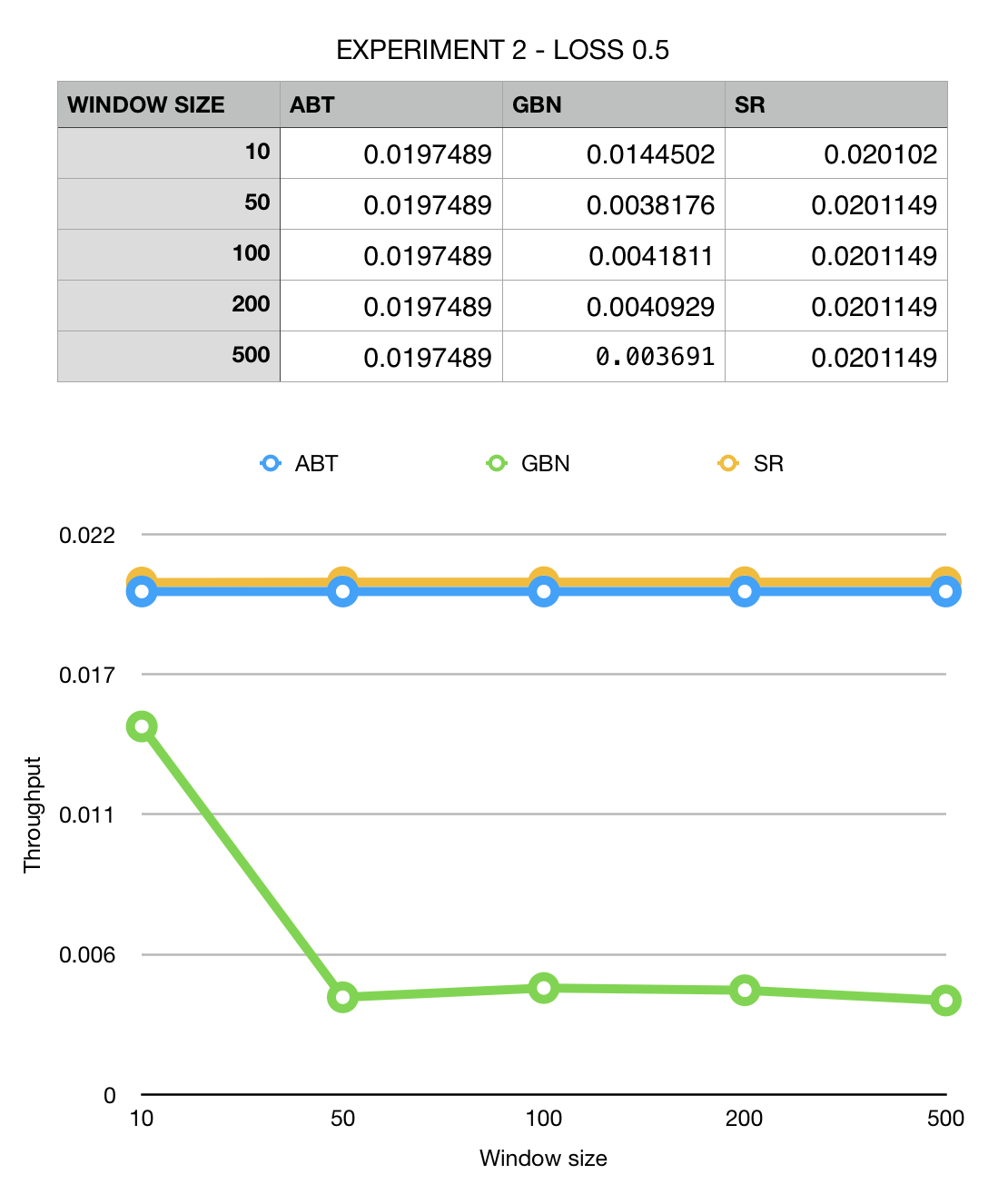
* All the three protocols have almost the same topology.
* For loss 0.8, SR drops steeply.

Experiment 2:



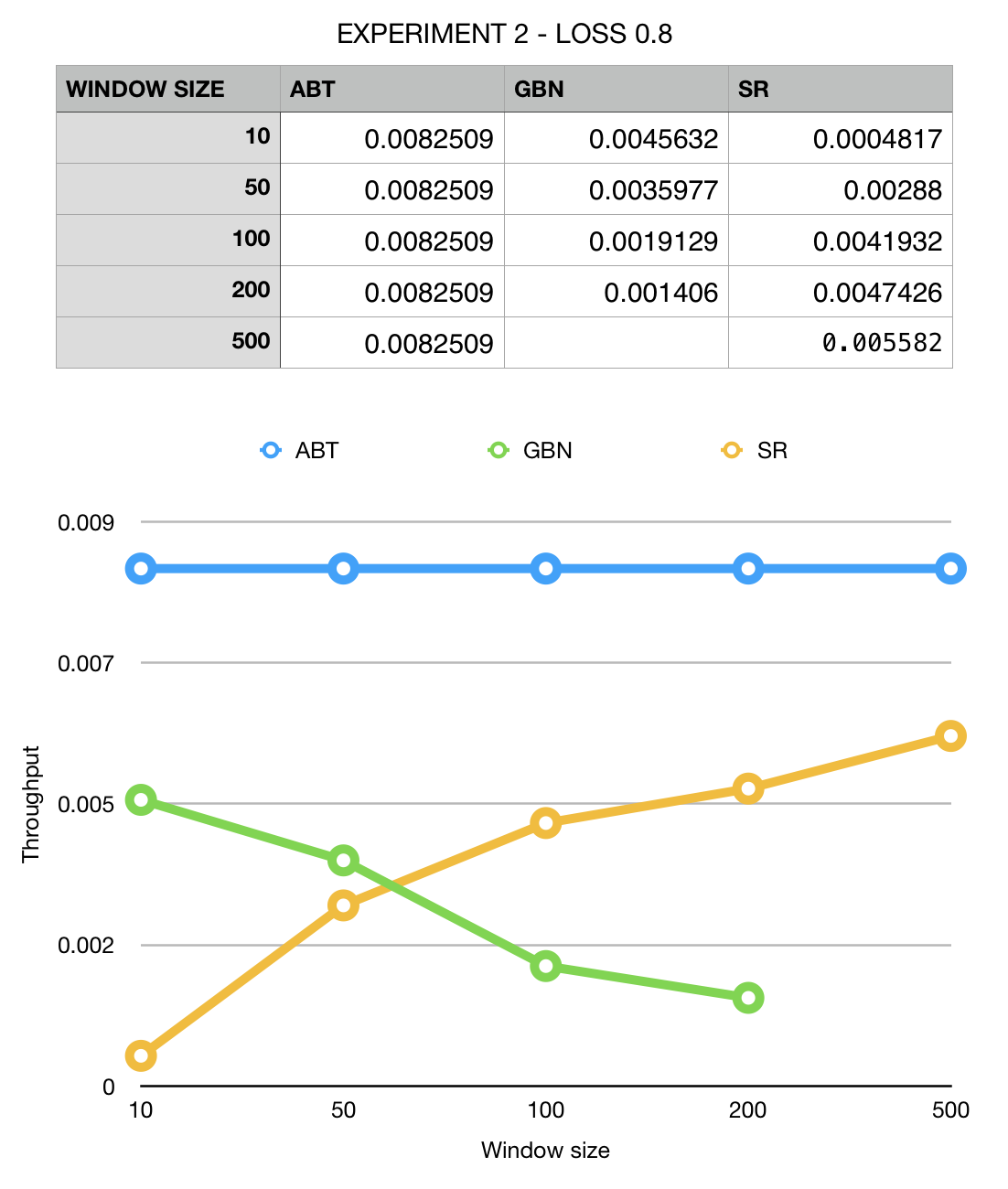
**Findings**

* SR gives good performance at all window sizes.
* GBN performs poorly compared to ABT, which is not an expected output.



**Findings**

* SR gives good performance at all window sizes.
* GBN performs poorly compared to ABT, which is not an expected output.



**Findings**

* SR gives good performance at all window sizes.
* GBN performs poorly compared to ABT, which is not an expected output.

**References:**

* Computer Networking - A Top Down Approach by Ross and Kurose
* Stackoverflow