

## CCN - Project 2

### Implementation and Simulation of Go-Back-N and Selective Repeat Protocols

Utkarsha Anil Gurjar- 801149356

Devika Unnikrishnan - 801135267

Sree Divya Keerthi Paravasthu Siddanthi - 801149985

#### Relevant Information

Additional classes used other than main sender and receiver :

- Ack: Its object is used to send information about acknowledgement.
- Checksum: It is used to tell if there are any checksum errors while sending the packets.
- Packet: It contains the details of each packet i.e. sent packet, error packet or last packet.

#### **Go-Back N**

The network designer or user selects a window size  $N$ . Typically,  $N$  is just large enough so that the pipe is full: the sender gets the acknowledgment of the first packet when it finishes transmitting packet number  $N$ . The sender numbers the packets sequentially modulo  $N + 1$ , that is, 1, 2, ...,  $N$ , 1, 2, ...,  $N$ , .... The receiver acknowledges every correct packet that it receives with an acknowledgment that it numbers as the largest numbered packet that it has received so far.

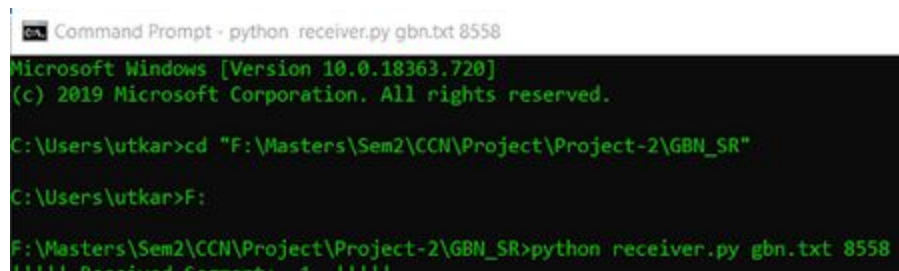
For instance, if the receiver gets the packets 1, 2, 4, 5, then it sends the acknowledgements with numbers 1, 2, 2, 2. Before transmitting packet number  $n + N$  (modulo  $N + 1$ ), the sender waits until it receives the acknowledgment of packet number  $n$ . Whenever the sender sends a packet, it starts a timer with a timeout  $T$  selected by the network designer.

When the sender fails to receive the acknowledgment of a packet before the packet times out, the sender retransmits that packet and all the packets that it has transmitted since it last transmitted that packet. Therefore, the receiver need not store any packet since it eventually gets them correctly in sequence.

The sender must be able to store up to  $N$  packets. Note that with this protocol the sender may have to retransmit packets that were correctly received by the receiver.

Here, we have implemented a Sender Receiver model in Java using UDP sockets and implemented Go Back N protocol on top of it as a flow control mechanism. The receiver takes the following command line arguments:

```
python receiver.py gbn.txt 8858
```

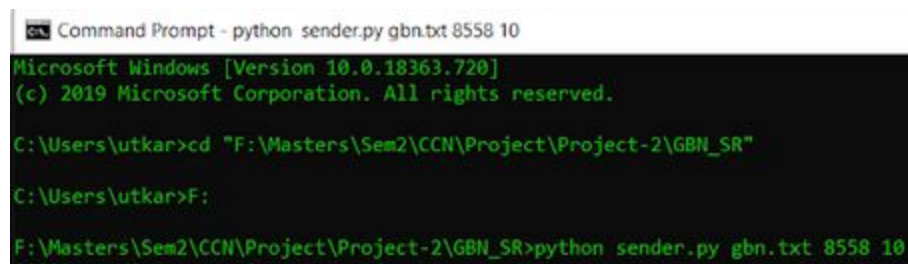


```
Command Prompt - python receiver.py gbn.txt 8558
Microsoft Windows [Version 10.0.18363.720]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\utkar>cd "F:\Masters\Sem2\CCN\Project\Project-2\GBN_SR"
C:\Users\utkar>F:
F:\Masters\Sem2\CCN\Project\Project-2\GBN_SR>python receiver.py gbn.txt 8558
!!! Received fragments: 1 !!!
```

Fig 1 - Receiver python file running

```
python sender.py gbn.txt 8858 10
```



```
Command Prompt - python sender.py gbn.txt 8558 10
Microsoft Windows [Version 10.0.18363.720]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\utkar>cd "F:\Masters\Sem2\CCN\Project\Project-2\GBN_SR"
C:\Users\utkar>F:
F:\Masters\Sem2\CCN\Project\Project-2\GBN_SR>python sender.py gbn.txt 8558 10
!!! Sending Time: 0.000000 !!!
```

Fig 2 - Sender python file running

## **Flow of Events -**

The total number of packets sent is 10 and window size is 7 .

1. Initially the sender sends the synchronization packet containing initial information like packet size, window size, Number of packets, Time out to the receiver.
2. The following are the screen shot for the Sender and Receiver, Initially we are sending 7 packets numbered 1 to 7. But on the receiver side the packet 4 had checksum error ( the data which the sender sent and the receiver received is different, so the check sum error occurs and the packets following packet 3 are discarded) so the receiver prints the “Checksum Error” message also as per the GBN protocol, and the following packets(Packet 4 to Packet 7) are discarded. For this the receiver prints the message “Packet Discarded” in the command line.
3. Here, since packet 4 is sent but has checksum error, the sender prints the timeout message in the command line. After that the sender will resend all the remaining packets in the window after Packet 3.
4. Also once the receiver correctly receives a particular packet the receiver sends the ACK for that packet, this is displayed in the sender window as a message “Received ACK”.
5. Also once the sender receives the ACK packets sent by the receiver the window is moved to accommodate the next set of packets. This is shown below in the screenshots, the sender receives ACK for packet 2 and packet 3 and 4 (timer for packet 4 is over), after that the window is moved and packet 4,5,6,7,8,9,10 is sent .
6. On the receiver side the packet 7 hasn't been reached so the receiver prints the “Packet 7 Lost” message also as per the GBN protocol, and the following packets(Packet 8,9,10) are discarded. For this the receiver prints the message “Discarding Packet” in the command line. It then receives ack for packet 8 which means packet 7 has been received after retransmission.

```
Command Prompt - python receiver.py gbn.txt 1234
Microsoft Windows [Version 10.0.18363.720]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\utkar>cd "F:\Masters\Sem2\CCN\Project\Project-2\GBN_SR"
C:\Users\utkar>F:
F:\Masters\Sem2\CCN\Project\Project-2\GBN_SR>python receiver.py gbn.txt 1234
!!!! Received Segment: 1 !!!!!
!!!! ACK Sent: 2 !!!!!
!!!! Received Segment: 2 !!!!!
!!!! ACK Sent: 3 !!!!!
!!!! Received Segment: 3 !!!!!
!!!! ACK Sent: 4 !!!!!
!!!! Checksum Error!!!! Packet: 4 discarded !!!!!
!!!! Packet: 5 lost :( !!!!!
!!!! Discarding packet: 6 !!!!!
!!!! ACK Sent: 4 !!!!!
!!!! Discarding packet: 7 !!!!!
!!!! ACK Sent: 4 !!!!!
!!!! Discarding packet: 4 !!!!!
!!!! ACK Sent: 4 !!!!!
!!!! Discarding packet: 5 !!!!!
!!!! ACK Sent: 4 !!!!!
!!!! Discarding packet: 6 !!!!!
!!!! ACK Sent: 4 !!!!!
!!!! Discarding packet: 7 !!!!!
!!!! ACK Sent: 4 !!!!!
!!!! Discarding packet: 8 !!!!!
!!!! ACK Sent: 4 !!!!!
!!!! Discarding packet: 9 !!!!!
!!!! ACK Sent: 4 !!!!!
!!!! Discarding packet: 10 !!!!!
!!!! ACK Sent: 4 !!!!!
!!!! Received Segment: 4 !!!!!
!!!! ACK Sent: 5 !!!!!

Command Prompt - python sender.py gbn.txt 1234 10
Microsoft Windows [Version 10.0.18363.720]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\utkar>cd "F:\Masters\Sem2\CCN\Project\Project-2\GBN_SR"
C:\Users\utkar>F:
F:\Masters\Sem2\CCN\Project\Project-2\GBN_SR>python sender.py gbn.txt 1234 10
!!!! Sending 1 ; Timer started.....
!!!! Sending 2 ; Timer started.....
!!!! Sending 3 ; Timer started.....
!!!! Sending 4 ; Timer started.....
!!!! Checksum wrong for packet: 4 !!!!!
!!!! Sending 5 ; Timer started.....
!!!! Sending 6 ; Timer started.....
!!!! Sending 7 ; Timer started.....
!!!! Received ACK: 2 !!!!!
!!!! Received ACK: 3 !!!!!
!!!! Received ACK: 4 !!!!!
!!!! Timeout segment: 5 , Resending....
!!!! Timeout segment: 6 , Resending....
!!!! Timeout segment: 7 , Resending....
!!!! Timeout segment: 4 , Resending....
!!!! Sending 4 ; Timer started.....
!!!! Sending 5 ; Timer started.....
!!!! Sending 6 ; Timer started.....
!!!! Sending 7 ; Timer started.....
!!!! Sending 8 ; Timer started.....
!!!! Sending 9 ; Timer started.....
!!!! Sending 10 ; Timer started.....
!!!! Timeout segment: 5 , Resending....
!!!! Timeout segment: 6 , Resending....
!!!! Timeout segment: 4 , Resending....
!!!! Sending 4 ; Timer started.....
!!!! Received ACK: 5 !!!!!
!!!! Timeout segment: 10 , Resending....
```

Fig 3 - Sender and Receiver

```
Command Prompt - python receiver.py gbn.txt 1234
!!!! Received Segment: 4 !!!!!
!!!! ACK Sent: 5 !!!!!
!!!! Received Segment: 5 !!!!!
!!!! ACK Sent: 6 !!!!!
!!!! Received Segment: 6 !!!!!
!!!! ACK Sent: 7 !!!!!
!!!! Packet: 7 lost :( !!!!!
!!!! Discarding packet: 8 !!!!!
!!!! ACK Sent: 7 !!!!!
!!!! Discarding packet: 9 !!!!!
!!!! ACK Sent: 7 !!!!!
!!!! Discarding packet: 10 !!!!!
!!!! ACK Sent: 7 !!!!!
!!!! Received Segment: 7 !!!!!
!!!! ACK Sent: 8 !!!!!
!!!! Received Segment: 8 !!!!!
!!!! ACK Sent: 9 !!!!!
!!!! Received Segment: 9 !!!!!
!!!! ACK Sent: 10 !!!!!
!!!! Received Segment: 10 !!!!!
!!!! ACK Sent: 11 !!!!!
!!!! Discarding packet: 10 !!!!!
!!!! ACK Sent: 11 !!!!!

Command Prompt - python sender.py gbn.txt 1234 10
!!!! Timeout segment: 10 , Resending....
!!!! Timeout segment: 8 , Resending....
!!!! Timeout segment: 9 , Resending....
!!!! Sending 5 ; Timer started.....
!!!! Timeout segment: 7 , Resending....
!!!! Sending 6 ; Timer started.....
!!!! Sending 7 ; Timer started.....
!!!! Sending 8 ; Timer started.....
!!!! Sending 9 ; Timer started.....
!!!! Sending 10 ; Timer started.....
!!!! Received ACK: 6 !!!!!
!!!! Received ACK: 7 !!!!!
!!!! Timeout segment: 7 , Resending....
!!!! Timeout segment: 8 , Resending....
!!!! Sending 7 ; Timer started.....
!!!! Received ACK: 8 !!!!!
!!!! Timeout segment: 9 , Resending....
!!!! Sending 8 ; Timer started.....
!!!! Sending 9 ; Timer started.....
!!!! Sending 10 ; Timer started.....
!!!! Timeout segment: 10 , Resending....
!!!! Received ACK: 9 !!!!!
!!!! Received ACK: 10 !!!!!
!!!! Sending 10 ; Timer started.....
!!!! Received ACK: 11 !!!!!
```

Fig 4 - Sender and Receiver continued

## SELECTIVE REPEAT:

### Introduction:

Selective repeat protocol is a data link layer protocol that uses a sliding window method for reliable delivery of data packets. Here, the only damaged or lost packets are retransmitted, while the good ones are received and buffered.

It uses two windows of equal size: a sending window which stores the packets to be sent and a receiving window which stores the packets received by the receiver. The size of the window is half the maximum sequence number.

### Working Principle

Selective Repeat protocol provides for sending multiple packets depending upon the availability of packets in the sending window, even if it does not receive acknowledgement for any packet in that process. The maximum number of packets that can be sent depends upon the size of the sending window.

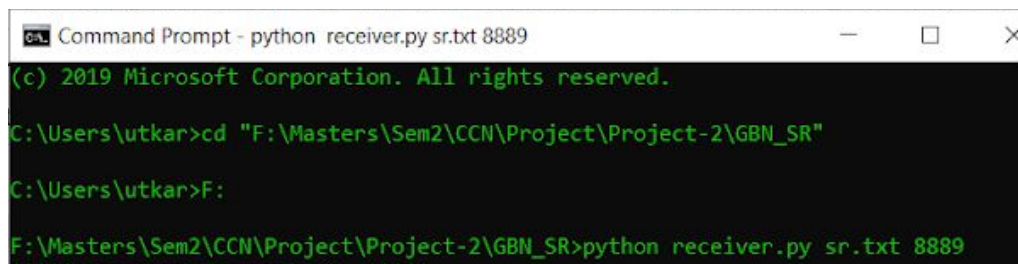
The receiver records the sequence number of the earliest erroneous or un-received packet. It then fills the receiving window with the subsequent packets that it has received. It sends the sequence number of the missing packet along with every acknowledgement packet.

The sender continues to send packets that are in its sending window. Once, it has sent all the packets in the window, it retransmits the packet whose sequence number is given by the acknowledgements. It then continues sending the other packets.

Here, we have implemented a Sender Receiver model in Python and implemented Selective Repeat protocol on top of it as a flow control mechanism.

For the receiver to work, we put the following command:

```
python receiver.py sr.txt 8889
```



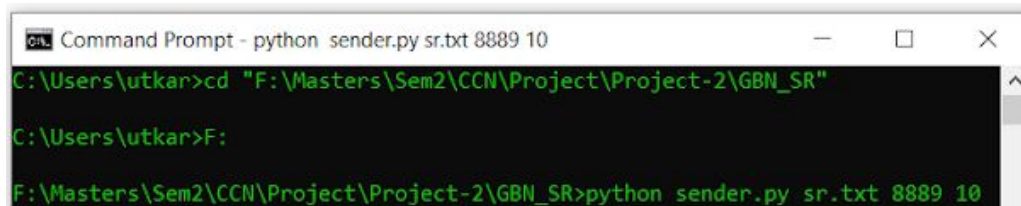
```
Command Prompt - python receiver.py sr.txt 8889
(c) 2019 Microsoft Corporation. All rights reserved.
C:\Users\utkar>cd "F:\Masters\Sem2\CCN\Project\Project-2\GBN_SR"
C:\Users\utkar>F:
F:\Masters\Sem2\CCN\Project\Project-2\GBN_SR>python receiver.py sr.txt 8889
```

Fig 5 - Receiver python file running



After inserting the details of the receiver, we have to put the details of the sender for the SR protocol to work.

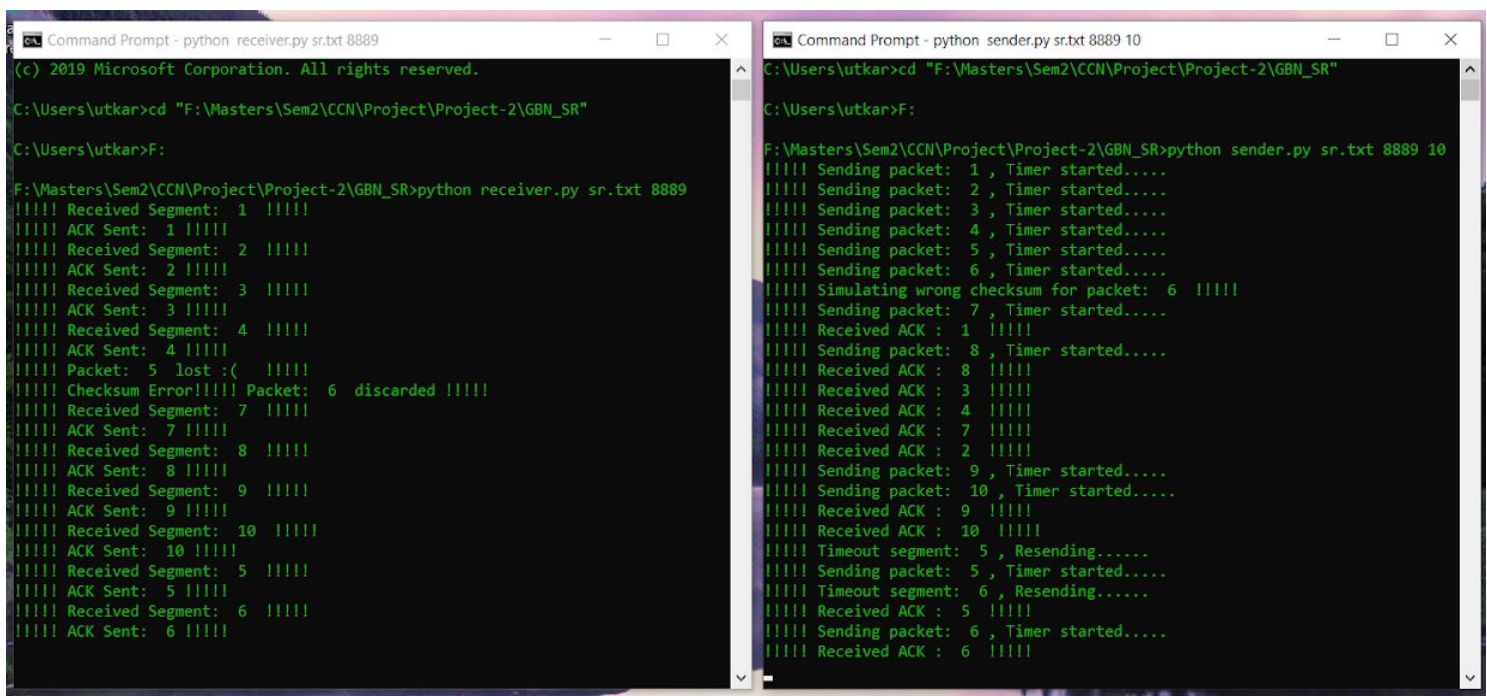
For the sender to work, we put the following command:  
python sender.py sr.txt 8889 10



```
Command Prompt - python sender.py sr.txt 8889 10
C:\Users\utkar>cd "F:\Masters\Sem2\CCN\Project\Project-2\GBN_SR"
C:\Users\utkar>F:
F:\Masters\Sem2\CCN\Project\Project-2\GBN_SR>python sender.py sr.txt 8889 10
```

Fig 6 - Sender python file running

## Working



```
Command Prompt - python receiver.py sr.txt 8889
(c) 2019 Microsoft Corporation. All rights reserved.
C:\Users\utkar>cd "F:\Masters\Sem2\CCN\Project\Project-2\GBN_SR"
C:\Users\utkar>F:
F:\Masters\Sem2\CCN\Project\Project-2\GBN_SR>python receiver.py sr.txt 8889
!!!! Received Segment: 1 !!!!!
!!!! ACK Sent: 1 !!!!!
!!!! Received Segment: 2 !!!!!
!!!! ACK Sent: 2 !!!!!
!!!! Received Segment: 3 !!!!!
!!!! ACK Sent: 3 !!!!!
!!!! Received Segment: 4 !!!!!
!!!! ACK Sent: 4 !!!!!
!!!! Packet: 5 lost :( !!!!!
!!!! Checksum Error!!!! Packet: 6 discarded !!!!!
!!!! Received Segment: 7 !!!!!
!!!! ACK Sent: 7 !!!!!
!!!! Received Segment: 8 !!!!!
!!!! ACK Sent: 8 !!!!!
!!!! Received Segment: 9 !!!!!
!!!! ACK Sent: 9 !!!!!
!!!! Received Segment: 10 !!!!!
!!!! ACK Sent: 10 !!!!!
!!!! Received Segment: 5 !!!!!
!!!! ACK Sent: 5 !!!!!
!!!! Received Segment: 6 !!!!!
!!!! ACK Sent: 6 !!!!!

Command Prompt - python sender.py sr.txt 8889 10
C:\Users\utkar>cd "F:\Masters\Sem2\CCN\Project\Project-2\GBN_SR"
C:\Users\utkar>F:
F:\Masters\Sem2\CCN\Project\Project-2\GBN_SR>python sender.py sr.txt 8889 10
!!!! Sending packet: 1 , Timer started....
!!!! Sending packet: 2 , Timer started....
!!!! Sending packet: 3 , Timer started....
!!!! Sending packet: 4 , Timer started....
!!!! Sending packet: 5 , Timer started....
!!!! Sending packet: 6 , Timer started....
!!!! Simulating wrong checksum for packet: 6 !!!!!
!!!! Sending packet: 7 , Timer started....
!!!! Received ACK : 1 !!!!!
!!!! Sending packet: 8 , Timer started....
!!!! Received ACK : 8 !!!!!
!!!! Received ACK : 3 !!!!!
!!!! Received ACK : 4 !!!!!
!!!! Received ACK : 7 !!!!!
!!!! Received ACK : 2 !!!!!
!!!! Sending packet: 9 , Timer started....
!!!! Sending packet: 10 , Timer started....
!!!! Received ACK : 9 !!!!!
!!!! Received ACK : 10 !!!!!
!!!! Timeout segment: 5 , Resending.....
!!!! Sending packet: 5 , Timer started....
!!!! Timeout segment: 6 , Resending.....
!!!! Received ACK : 5 !!!!!
!!!! Sending packet: 6 , Timer started....
!!!! Received ACK : 6 !!!!!
```

Fig 7 - Sender- Receiver running python file

1. Initially the sender sends the synchronization packet containing initial information like the sender python file; the sr document containing the details like the protocol used, sequence bits, window size, timeout period etc; port number and number of bits to be sent as shown in fig(6).

2. After connecting the sender, we have to make the connection of the receiver. Here, we have to put the same port number as we put in the sender for connection purposes. As shown in fig(5), we make the receiver connection. It contains information like the receiver python file; the sr document containing the details like the protocol used, sequence bits, window size, timeout period etc and the port number.

3. We are setting the window size as 6 (It is mentioned in the sr.txt file) and the number of packets as 10 (to be put in the command line argument).

4. As shown in fig(7), the sender sends the first 6 packets and the timer starts as soon as it is sent. There arises checksum error for packet 6. The sender keeps sending the other packets. Parallely, the receiver is sending the acknowledgements of the packets which are received.

5. Here, at the receiver side packet 5 is lost and checksum error at packet 6. So the sender sends only these packets to the receiver which has some ambiguity.

6. At the sender side, there is a timeout for the packets 5 and 6. So, the sender sends only these packets to the receiver.

7. Thus, in SR protocol, only the lost or the damaged packets are sent again by the sender.