Project 2 – Report

Agasthya Vidyanath Rao Peggerla

Programming Language used: JAVA

**UDP:** UDP uses a simple connectionless transmission model with a minimum of protocol mechanism. UDP provides checksums for data integrity, and port numbers for addressing different functions at the source and destination of the datagram. It has no handshaking dialogues, and thus exposes the user's program to any unreliability of the underlying network: there is no guarantee of delivery, ordering, or duplicate protection.

UDP is suitable for purposes where error checking and correction are either not necessary or are performed in the application; UDP avoids the overhead of such processing at the level of the network interface. Time-sensitive applications often use UDP because dropping packets is preferable to waiting for delayed packets, which may not be an option in a real-time system.

**Sliding Window Protocol:** A sliding window protocol is a feature of packet-based data transmission protocols. Sliding window protocols are used where reliable in-order delivery of packets is required, such as in the Data Link Layer (OSI model) as well as in the Transmission Control Protocol (TCP).

Conceptually, each portion of the transmission (packets in most data link layers, but bytes in TCP) is assigned a unique consecutive sequence number, and the receiver uses the numbers to place received packets in the correct order, discarding duplicate packets and identifying missing ones. The problem with this is that there is no limit on the size of the sequence number that can be required.

**Go-Back-N (GBN):** Go-Back-N ARQ is a specific instance of the automatic repeat request (ARQ) protocol, in which the sending process continues to send several frames specified by a window size even without receiving an acknowledgement (ACK) packet from the receiver. It is a special case of the general sliding window protocol with the transmit window size of N and receive window size of 1. It can transmit N frames to the peer before requiring an ACK.

The receiver process keeps track of the sequence number of the next frame it expects to receive, and sends that number with every ACK it sends. The receiver will discard any frame that does not have the exact sequence number it expects (either a duplicate frame it already acknowledged, or an out-of-order frame it expects to receive later) and will resend an ACK for the last correct in-order frame.[1] Once the sender has sent all of the frames in its window, it will detect that all of the frames since the first lost frame are outstanding, and will go back to the sequence number of the last ACK it received from the receiver process and fill its window starting with that frame and continue the process over again.

**Selective Repeat:** Selective Repeat is part of the automatic repeat-request (ARQ). With selective repeat, the sender sends several frames specified by a window size even without the need to wait for individual ACK from the receiver as in Go-Back-N ARQ. The receiver may selectively reject a single frame, which may be retransmitted alone; this contrasts with other forms of ARQ, which must send every frame from that point again. The receiver accepts out-of-order frames and buffers them. The sender individually retransmits frames that have timed out.

It may be used as a protocol for the delivery and acknowledgement of message units, or it may be used as a protocol for the delivery of subdivided message sub-units.

When used as the protocol for the delivery of messages, the sending process continues to send several frames specified by a window size even after a frame loss. Unlike Go-Back-N ARQ, the receiving process will continue to accept and acknowledge frames sent after an initial error; this is the general case of the sliding window protocol with both transmit and receive window sizes greater than 1.

**Structure of Code:**

There are 4 classes:

1. Packet
2. AckPacket
3. SenderProcess
4. ReceiverProcess

**Packet:** The packet class represents the packet that is to be sent from the sender to the receiver. The packet class three attributes: sequence number, data and checksum. The methods in this class are:

* setSnum – this method is used to set the sequence number of the packet
* getSnum – this method is used to get the sequence number of the packet
* setData – this method is used to initialize the data in the packet
* getData – this method is used to retrieve the data in the packet
* setChecksum – this method is used to set the checksum of the packet
* getChecksum – this method is used to get the checksum of the packet

**AckPacket:** The ackpacket class represents the ACKPACKET that is to be sent from the receiver to the sender. The packet class has sequence number as an attribute. The methods in this class are:

* setSnum – this method is used to set the sequence number of the packet
* getSnum – this method is used to get the sequence number of the packet

**SenderProcess:** This class contains the sender side process to send the data to the receiver. We use three arraylists to store the information about the packets- one which has all the packets, one which has record of all the sent packets, and one which has information about the received acknowledgements. The methods in this class are:

* main() – this is the main method in the class which has the implementation of the protocols.
* cal\_checksum – this method is used to calculate the checksum of the packets.
* make\_packet – this method is used to make packets by randomly generating data.

**ReceiverProcess:** This class contains the receiver side process to receive data and send acknowledgements to the sender upon receiving data. We use arraylists to store the information about packets- one to save all the received packets and one to save the acks sent to the sender upon receiving the data. The methods in this class are:

* main() – this is the main method in the class which has the implementation to send acks after the packets have been received.
* make\_AckPacket – this method is used to initialize the ack packets.
* cal\_checksum – this method is used to calculate the checksum of the packets.

**Execution Steps:**

1. Open command prompt where the JAVA files are present.
2. Run ReceiverProcess.java by passing PORTNO, Protocol name as command line argument.
3. Run SenderProcess.java by passing Input\_filename, PORTNO, No of packets to be sent as command line argument.

Note: The input.txt file was used by us. Which can be used to set the protocol to be used, no of bytes for sequence numbers, window size, timeout and MSS.

For changing the protocols, you need to change the protocol name in the input.txt file.

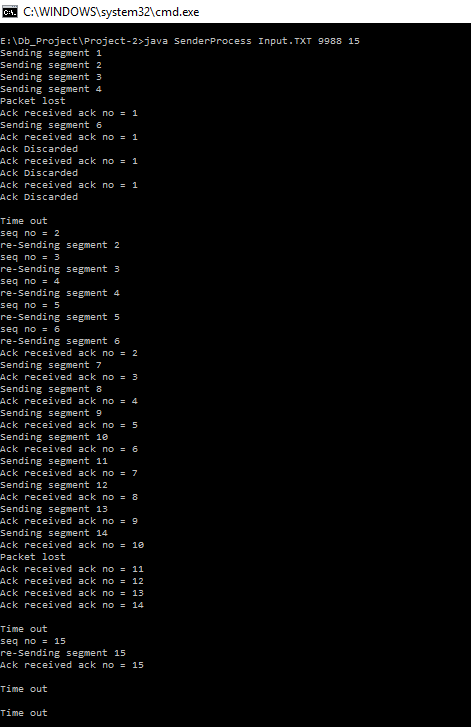
Example:

* java ReceiverProcess 8000 GBN/SR
* java SenderProcess input.txt 8000 15

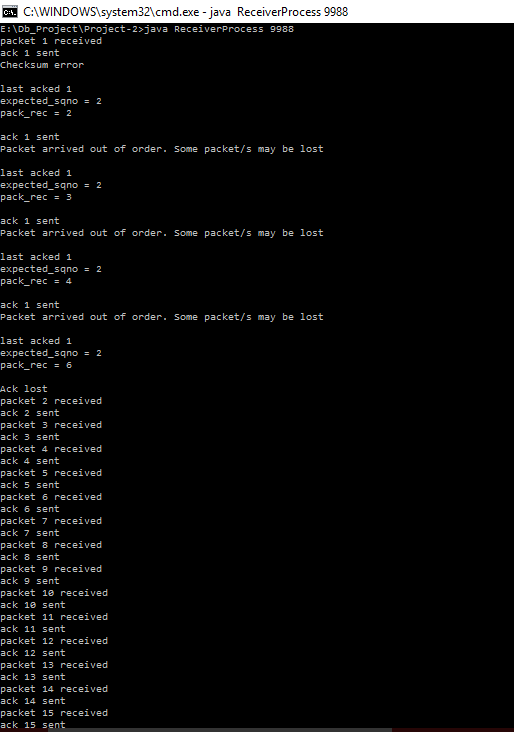
Sample Outputs:

**Go-Back-N:**

**Sender-**

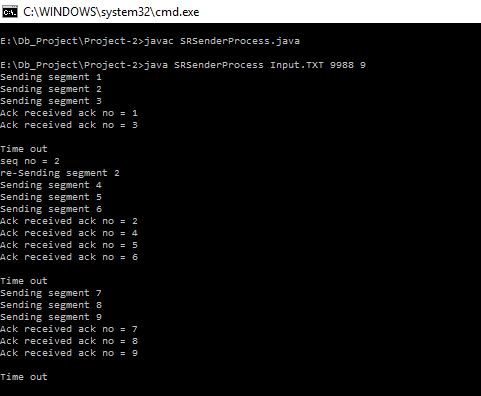
****

**Receiver-**

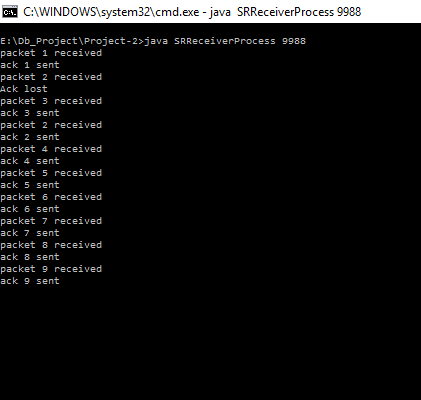
****

**Selective Repeat:**

**Sender-**

****

**Receiver-**

****