COMP 7005
Final Project Design Document

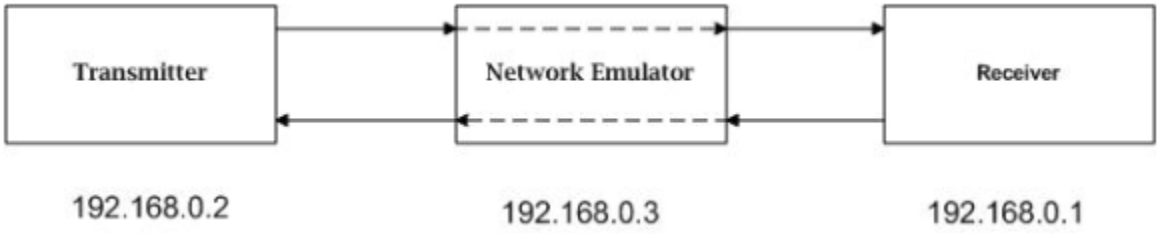
By: Derek Wong and Maksym Chumak

Instructor: Aman Abdulla

Due: 11 PM on December 4th, 2020

Overview

Implement a Send-And-Wait protocol simulator that will be half-duplex and uses sliding windows to send multiple packets between hosts on a LAN with an “unreliable network” between two hosts. The following diagram depicts the model:



Requirements

The application has the following constraints:

1. Any language of choice to implement the three components shown in diagram above
2. Design application layer protocol, on top of UDP
3. Protocol will be half-duplex and use sliding windows to send multiple packets between two hosts
    a. One side will be allowed to acquire the channel first and send packets
    b. Other side will have to wait until the other side has completed sending packets before sending its own packets
    c. Window will slide forward with each ACK received, until all frames in current window have been ACKed
4. Protocol should be able to handle network errors such as packet loss and duplicate packets
5. Use timeouts and ACKs to handle retransmissions to lost packets, based on the automatic repeat request (ARQ) error control mechanisms
    a. Implement a timer to wait for ACKs or to initiate a retransmission in case of a no response for each frame in the window
6. Network emulator will act as unreliable channel over which the packets will be sent, relaying packets from the transmitting to the receiver and from the receiver to the transmitter
7. Network emulator will include a “noise” component randomly discarding all types of packets to achieve a specific bit error rate
8. Application architecture should have a minimum of three source modules:
    a. Transmitter
    b. Receiver
    c. Network
    d. Associated include files and libraries
9. Bit error rate, average delay per packet, endpoint addresses/ports of transmitter, sender and network emulator should be extracted from common configuration file
10. Transmitter will provide end of transmission signal once all the packets are sent
11. Both transmitter and receiver will print out the ongoing session as simple text lines containing the type of packets sent, type of packet received, status of the window, sequence numbers
12. Application will maintain a log file at both the transmitter and receiver. Use this to troubleshoot and validate your protocol

Design

- The transmitter/receiver application will be written in C
    - Use of Berkeley Sockets API
- The network emulator will be written in C++
    - Use of QT's QUdpSocket
- Shared logic, logging, packet implementation and configuration constants(Bit Error Rate, average delay per packet, endpoint addresses/ports of transmitter, sender and network emulator) will be imported from the shared header files
- Packets Information will be stored in a C struct:

```
struct packet
{
        int packetType;
        int seqNum;
        char data[payloadLen]
        int windowSize;
        int ackNum;
        int retransmit
};
```

- Packets can have the following types:

```
enum PacketType { DATA, ACK, EOT };
```

- Logging in the format:

```
[<severity level (DEBUG, INFO, ERROR)>][<date time>] <message>
{
        <packet struct key>: <packet struct value>,
}
```

Transmitter

1. Connects to network using specified address or default address from the configuration file
2. Stores and sends a file using specified file path or default path from the configuration file
    a. Each line in file is stored in a packet struct
    b. When sent to the network emulator, packetType is set to DATA
    c. Each line will correspond to a different seqNum
    d. Ack num will be set to null
    e. Retransmit is only set to true in retransmissions
3. Transmitter will have the following states:
    enum State { SendingPackets, WaitForACKs, AllACKsReceiverd, AllPacketsSent };
4. Upon receiving ACKs and packet loss events update the timeout interval value
5. Timeouts Intervals will be calculated by the following formula:
    a. TimeoutInterval = EstimatedRTT + 4*DevRTT, where
        i. EstimatedRTT = (1- α)*EstimatedRTT + α*SampleRTT (α = 0.125)
        ii. DevRTT = (1-β)*DevRTT + β*|SampleRTT-EstimatedRTT| (β = 0.25)
6. Transmitter will selectively retransmit packets when the round trip time of the previous window exceeds the timeout interval
    a. Packets include those that have not been ACKed yet, and will not send a new window of packets until all packets in previous window has been ACKed
7. 10 EOT packets will be sent when AllPacketsSent state is achieved to reliably signal to the receiver to terminate

Network Emulator

1. Display of relayed packets including packet loss events and retransmissions in table
2. Display of network summary including number of relayed packets, dropped packets, retransmitted packets and total capture time
3. Display Time Sequence diagram of packets sent from transmitter to network emulator
4. Configurable in real time Bit Error rate and network delay using slider
5. Display of Network Emulator application status(Active/Stopped)
6. Reset UI to the initial state functionality
7. Stop application functionality

Receiver

1. Receiver listens for packets on the configured port:
    a. upon receiving an in order DATA packet the receiver saves the data to the file and responds with ACK packet, acknowledgment number of the packet matches the sequence number of the corresponding DATA packet
    b. upon receiving an out of order packet the receiver buffers the packet and responds with ACK packet, acknowledgment number of the packet matches the sequence number of the corresponding DATA packet
    c. upon receiving a new window of packets all data from the buffer is written to the file
    d. upon receiving a EOT packet the receiver application terminates

Protocol Design

Sender

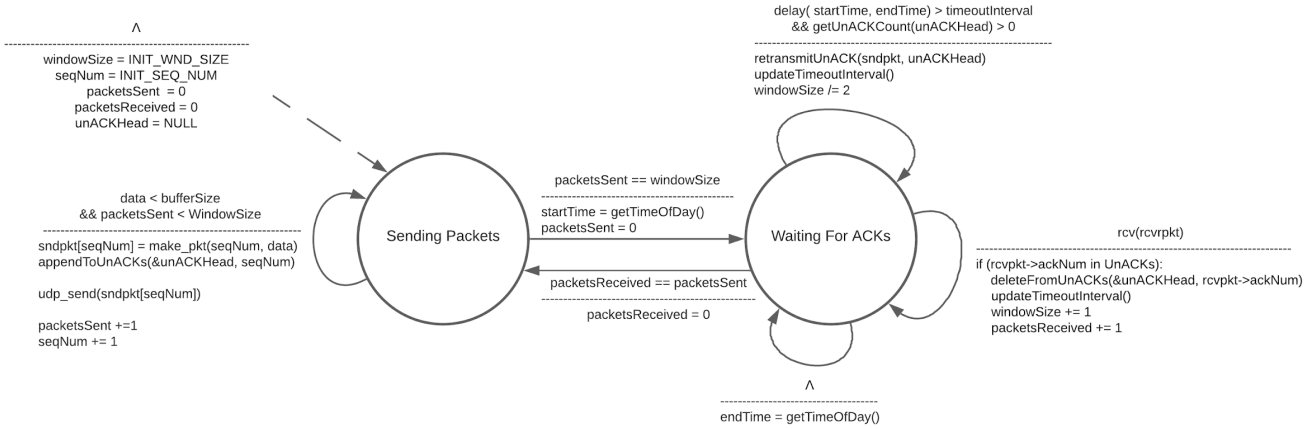


Figure 1 – sender finite state diagram(protocol).

Unreliable Network

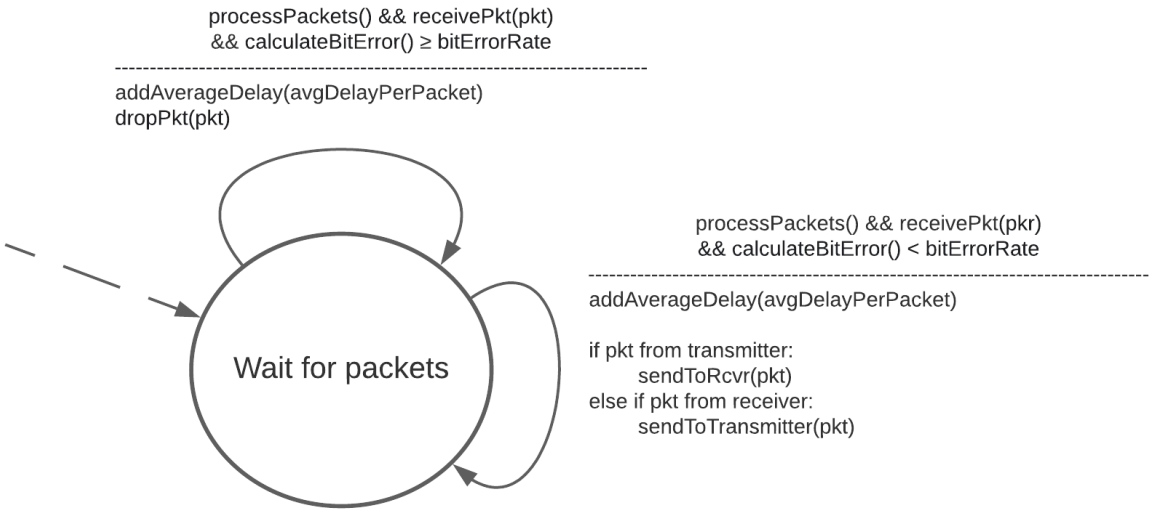


Figure 2 – network emulator finite state diagram(protocol).

Receiver

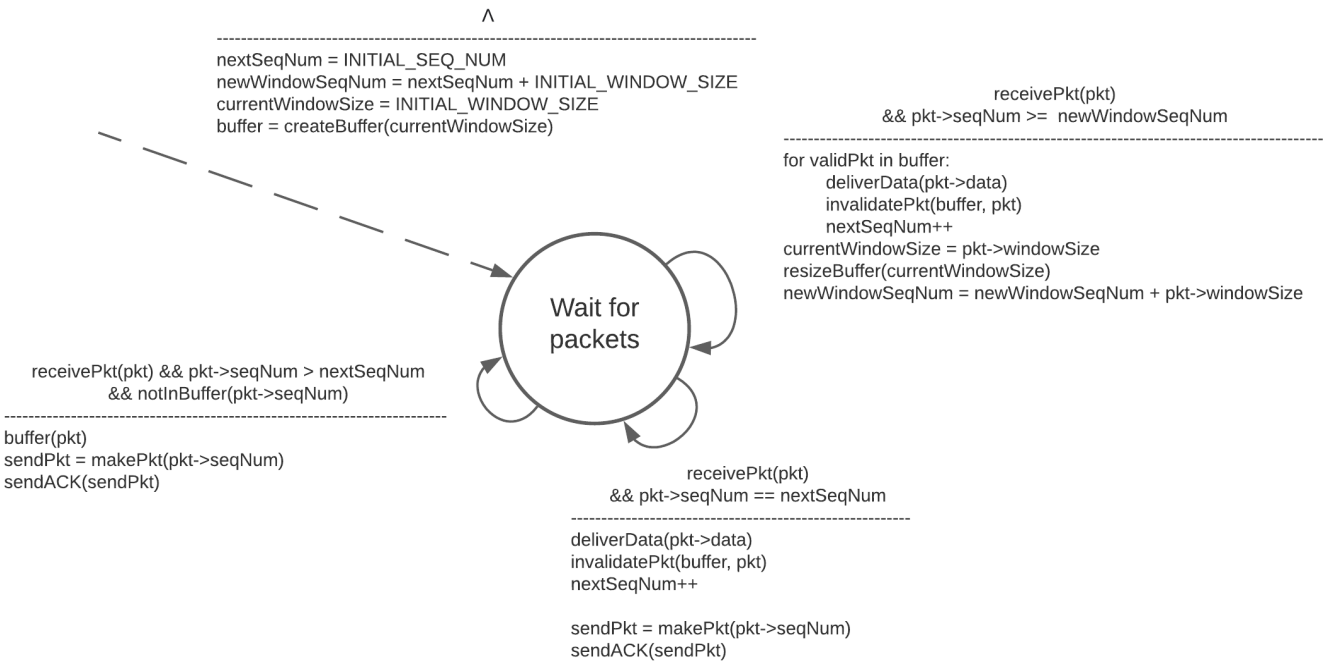


Figure 3 – receiver finite state diagram(protocol).

Application Design

Transmitter

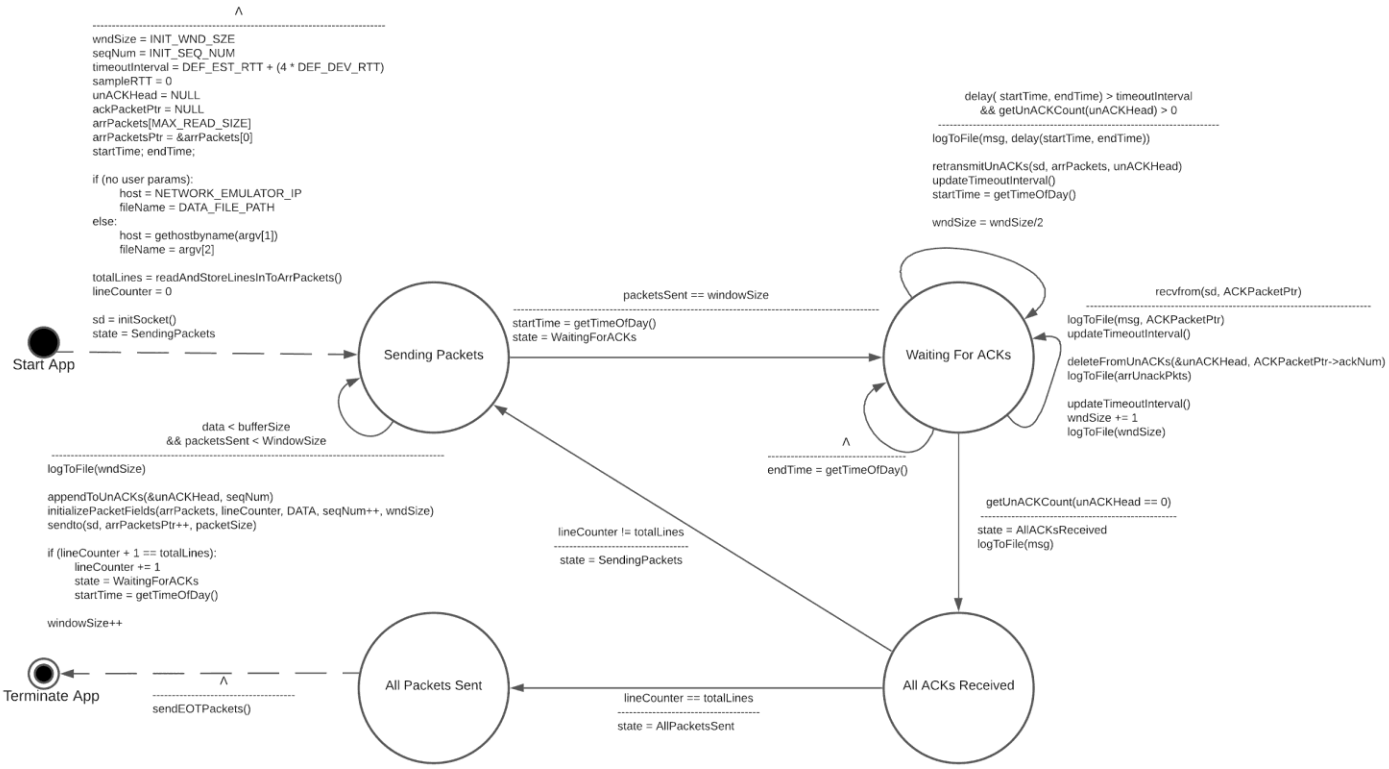


Figure 4 – transmitter finite state diagram(application).

Network Emulator

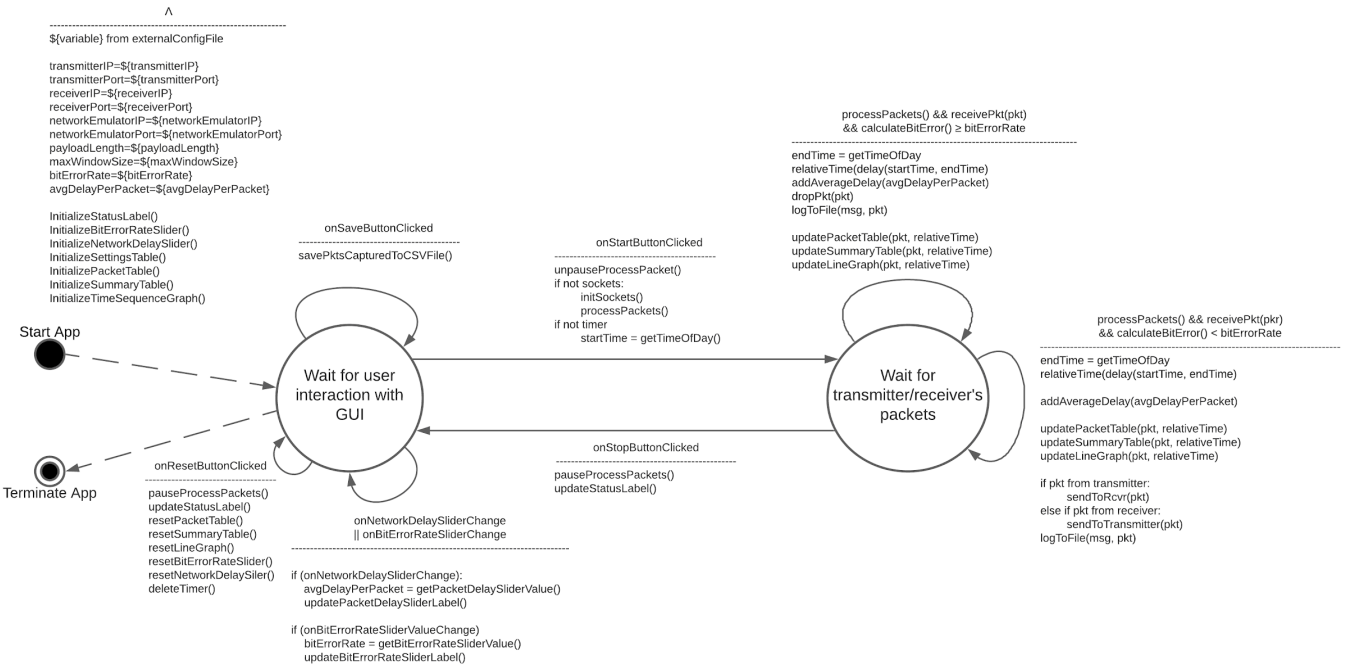


Figure 5 – network emulator finite state diagram(application).

Receiver

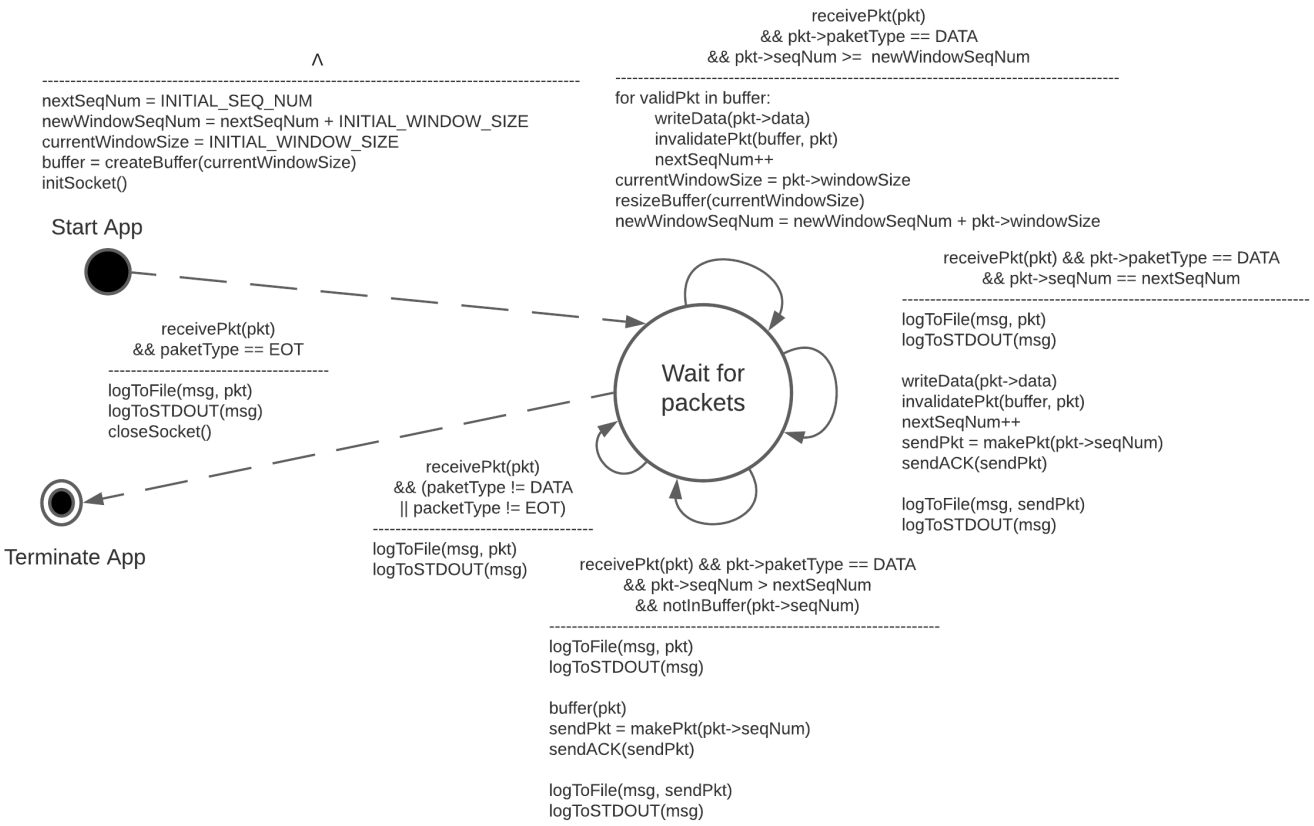


Figure 6 – receiver finite state diagram(application).

Pseudo Code

Transmitter

Initialize windowSize, seqNum, timeoutInterval, estimatedRTT, devRTT, readTimeout from constants from header file

Initialize array of packets (structs) with length determined by constant
Declare arrPacketsPtr to iterate over array of packets
Declare node* unACKHead pointer to a linked list of unACKed packets
Initialize start and end time to track transmission progress

Declare enumerator of transmitter states
{ SendingPackets, WaitForAcks, AllAcksReceived, AllPacketsSent }

Declare enumerator of packet types
{ DATA, ACK, EOT }

Get User Parameters from command line
Check if user specifies network emulator IP or file to send, then set as networkEmulatorIP/fileName

Create UDP socket
If failed to create a UDP socket
Log error
Exit the program

Set socket options for UDP socket for non-blocking mode
Bind socket to transmitter port
If failed to bound
    Log error
    Exit the program

Open file and iterate through file contents:
For line in file:
    Add to array of packets (structs)
Log number of lines

Initialize and define totalLines and store number of lines in file
Initialize lineCounter to 0

```
Set initial state to SendingPackets
While state is not AllPacketsSent
        If state is SendingPackets
        For each packet in window size:
                Set packetType to DATA;
                Set seqNum++
                Set windowSize
                Set ackNum to be null
                Set retransmit to be false

Send packet to receiver
Add to array of UnACKed packets
                If lineCounter + 1 != totalLines
                        lineCounter++
                        Set transmitter state to WaitForACKs
                        Start timer
        Log state change to Waiting for acks
If state is WaitForAcks
        If the number of unACKs is 0
                Log all acks received
                Set state to AllACKsReceived
        Record time since timer started
        Log the delay for round trip time of packets
        If delay is greater than timeout interval and there remains unACKs in unACK array
                Log that RTT is longer than timeout interval
                Retransmit all packets in unACK array
                Update TimeoutInterval
                Reset timer
                Reduce window size by half
                Receive ACKs from network emulator
                Log ACK num received
                Update TimeoutInterval
                If data from receiver contains ACK we haven't received yet:
                        Remove from unACKed list
                        Increase window size by one
        If state is AllACKsReceived
                Set state to AllPacketsSent if lineCounter == totalLines else to SendingPackets
Log completed data transfer
Log sending EOT packet
Make 10 EOT for reliable delivery of packets to the network emulator (some may drop)
Terminate transmitter
Log terminating transmitter
Deallocate dynamically allocated memory
```

Receiver

Declare initial sequence number constant and set it to 1
Declare initial window size constant and set it to 1
Declarare invalid sequence number constant and set it to 0
Declare enumerator of packet types { DATA, ACK, EOT }
Declare expected packet structure

Create UDP socket
If failed to create a UDP socket
Log error
Exit the program

Bind socket to receiver port
If failed to bound
    Log error
    Exit the program

Set sequence number of the next in order packet to initial sequence number
Set sequence number of the first packet in the next window to initial sequence number + initial window size
Set current window size to initial window size

Dynamically allocate array of packets of initial window size
Set each packet's number in the array to invalid seq num

```
Loop
        Receive packet
If socket received an invalid packet
Log error
Exit the program
If packet type is DATA
        // new window
If packet sequence number >= new window size
        Iterate over unsaved buffered out of order packets
                If packet's sequence number is not invalid
                Save packet's data to a file
                Set packet's sequence number to invalid
                Increment expected in order sequence number by 1
Set current window size to packet's window size
Set  sequence number of the first packet in the next window to its current value + packet's window size
Reset dynamically allocated array of packets to the new window size
        Log received DATA packet details
        If packet's sequence number == sequence number of the next in order packet
                Save packet data to a file
                Set packet's sequence number in the buffer to invalid
                Increment expected in order sequence number by 1
                Else if packet's sequence number > in order sequence number and packet
not in the buffer
Add packet to the buffer
Make ACK packet with acknowledgement number equal to received packet's sequence number
Send to the transmitter's address
Log ACK packet details
Else if packet type is EOT
        Log packet details
        Log final message
        Terminate the program
Else
        Log packet details
```

Network Emulator

```
Set window title
Initialize status label
Initialize Bit Error Rate slider
Initialize network delay slider
Initialize packet table
Initialize summary table
Initialize sequence time graph
If start button is clicked
        Set pause to false
        If timer not initialized
                Initialize timer
        If UDP socket does not exist
                Create UDP socket
                Bind socket to configured IP and port
                Loop
                        Receive packet
                        If pause == true
                                Go to beginning of the loop
                        If valid IP and valid port
                                Apply network delay
                                If packet not dropped
                                If packet received from transmitter
                                        Send packet to receiver
                                        Update time sequence graph
                                        Update packet table
                                Else if packet received from receiver
                                        Send packet to transmitter
                                        Update packet table
                                        Log packet details
                                Else
                                        Log unknown client message
                                Else
                                If packet received from transmitter
                                        Update time sequence graph
                                        Update packet table
                                Else if packet received from receiver
                                        Update packet table
                                        Log packet details
                                Update summary table
```

If stop button is clicked
    Set pause to true
    Set status label color to red
    Set status label text to “Stopped”
If reset button is clicked
    Reset packet table data
    Reset summary table data
    Reset time sequence graph
    Reset Bit Error rate slider to default value
    Reset network delay slider to default value
If save button is clicked
    Open file location prompt window
    Save packet table details as to .CSV file
If Bit Error Rate slider value was changed
    Set bit error rate value to slider’s value
    Update bit error rate slider label to match the value
If network delay slider value was changed
    Set network delay value to slider’s value
    Update network delay slider label to match the value
If close button is clicked
    Terminate the application

Implementation Analysis

Selective repeat and Timeouts

We chose to use the pipelined protocol, **selective repeat**, as a starting point to implement a reliable data transfer protocol. The receiver will individually acknowledge all correctly received packets, buffering packets as needed for eventual in-order delivery to the upper layer. The sender only resends packets for which ACKs are not received; however, instead of having a timer for each unACKed packet, the transmitter only keeps track of when the last packet in a window is sent. Another difference between our implementation is that the sender window does not slide immediately to the next unACKed packet on each ACK, since we are enforcing a half-duplex send and wait protocol -- All packets must be ACKed before sliding the window forward.

Our choice to not introduce a timer for each unACKed packet was to reduce complexity and CPU resource consumption required to find the delay between each individual packet sent in a burst and when the return from the receiver. The timing difference between packets sent in a window is negligible compared to the real network delay and our enforced per packet delay on the network emulator. We chose to use the same timeout interval calculation as TCP, which attempts to balance having too short or too long of a timeout. Using an exponential weighted moving average helped reduce the influence of past samples exponentially fast. This in turn reduces having premature timeouts, unnecessary retransmission or slow reactions to packet loss events. To this end, we observed that updating the timeout interval after each ACK and timeout events reduced the likelihood of premature timeouts but did not recover fast enough from packet loss events. Thus, it was necessary to use a max timeout value to prevent the interval from growing out of proportion from repeated packet loss events.

Seq & ACK number relation

We chose to increment the sequence number by one for each DATA packet that was sent by the transmitter starting with the initial value of 1; the acknowledgement number of the ACK packets returned by the receiver matches the sequence number of the acknowledged packet. Simple, additive increase in sequence numbers allowed to greatly simplify handling of the out of order packets on the receiver's side as well as made verifying protocol's functionality easier, since it's easy to spot DATA/ACK packet pairs. Moreover, we set a maximum window size value to limit the exponential growth of the window size. This helped reduce the amount of single threaded hanging of the network emulator UI based on our implementation of average packet delay.

Average packet delay

Initially **usleep** function was applied to delay each packet by an average of user provided value of milliseconds, however we noticed that frequent suspensions in program execution caused delays in UI with the increase in number of received packets. To address the problem, the time of packet arrival was compared to the current time, until the difference exceeds the delay value set by the user. A slider was introduced in the network emulator UI to allow users to control the delay value.

Bit Error Rate

Bit Error Rate was implemented using **rand** function, which generates a random number in the specified range. If the generated value was less than the specified BER, the packet was dropped. A slider was introduced in the network emulator UI to allow users to control how often packets would be dropped. For example, once the bit error rate is set to 100%, every single packet being sent by either the transmitter or receiver will be dropped and never relayed to the other machine.

Conclusion

Based on our implementation of half-duplex protocol simulator with sliding windows and with modified selective repeat (ARQ) error-controls mechanism, we were able to send and receive a text file reliably and successfully over an unreliable network. Furthermore, we were able to capture traffic and analyze how the modification to bit error rate and network delay affects the packet loss events. With this, we were able to display the behavior of both machines using a network time-sequence graph, packet capture and network summary statistics table.