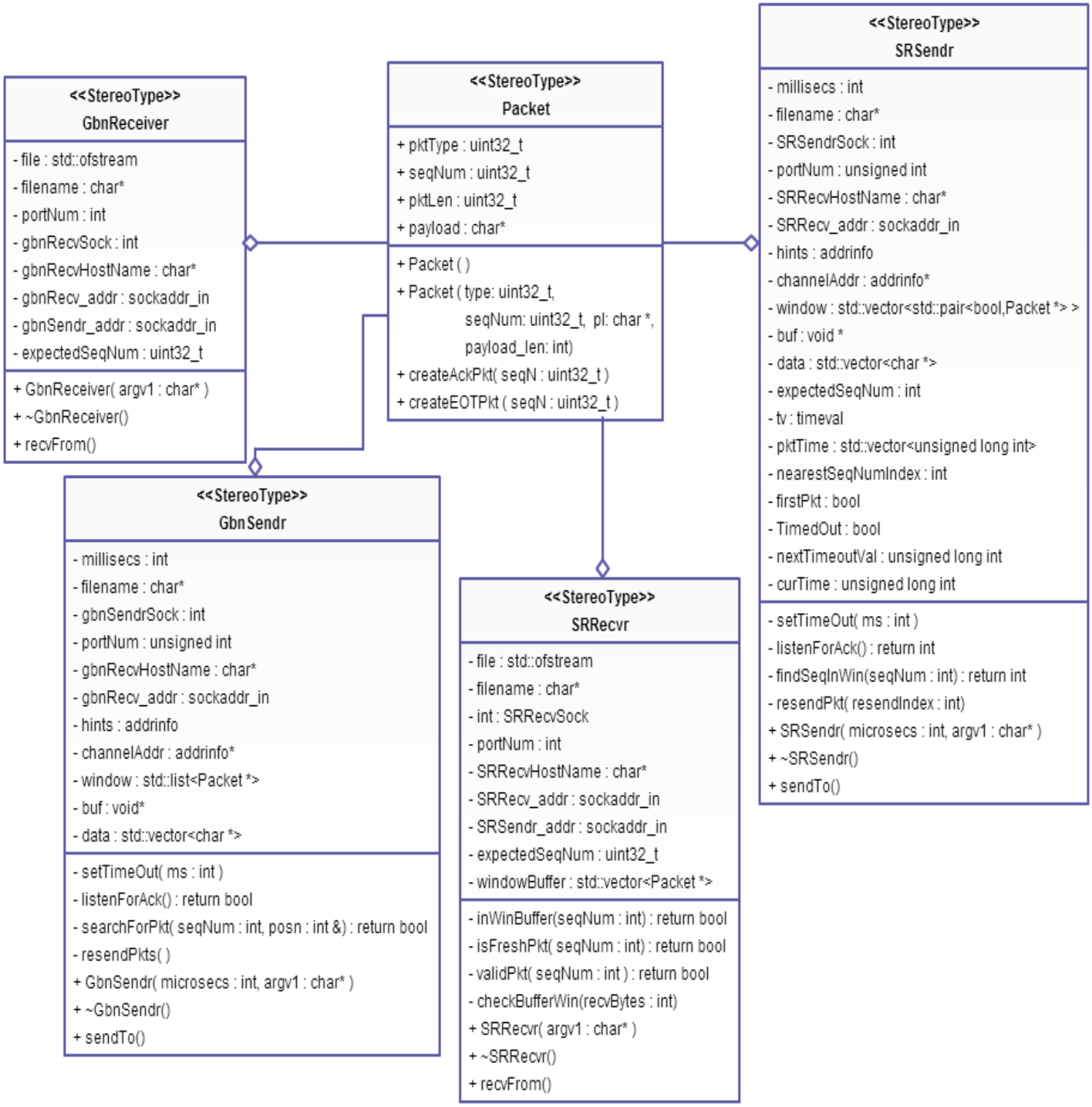

Class Diagram for the GBN and SR protocol implementation

Jayanth Kottapalli
20347318 - jkottapa
CS456 - Assignment 1

Abstract

This documentation is to describe the design and testing of the implementation for the Go-Back-N and Selective Repeat reliable pipelined transfer protocols. The design for each of the protocols will be described in detail including any design justifications as well as reasoning behind the optimizations that were used for each of the implementation. A brief overview of how each of the protocols' implementation is designed will be provided before we divulge into the intricate details. Each of the classes will be outlined with all the necessary details along with their respective purposes for the protocols. How each of the implementations were tested will be explained once the design has been thoroughly detailed

Packet class

This class consists of the details of the packets sent via the sockets to the both the sender and the receiver. Both versions of protocols use multiple Packet class objects to transfer the data in the files over from the sender to the receiver. As illustrated in the above class diagram, a Packet class consists of 4 main details corresponding of each packet sent over the socket. Three of the variables will contain the information regarding the packet's header such as the Packet Type, the Sequence Number and the Packet Length. The last variable will contain the Payload or the data, as sent by the sender or the receiver. The packet header details are converted to network byte order type, while the payload will remain as a sequence of characters that are provided to the class.

There are two overloaded constructors provided in the Packet class to create an empty packet or a complete packet with all the information. There are two additional functions provided by this class to assist in creating specific type of packets.
Purpose: To create an acknowledgement packet with just the header information and no payload data of the given sequence number
+ createAckPkt(...)
Purpose: To create an EOT packet with just the header information and no payload data of the given sequence number
+ createEOTPkt(...)

Go-Back-N

The design of the go-back-N protocol is divided across two classes called GbnReceiver and GbnSendr, where the GbnReceiver class is implemented to perform as the receiver for the go-back-n protocol while GbnSendr class is implemented to perform as the sender for the go-back-n protocol. As described in the assignment specification, the sender sends packets to the channel emulator with a connection established via a socket whose details are retrieved from the "channelInfo" file. The receiver requests for a port number to bind itself to and opens a socket, in order to listen to any packets forwarded to it by the channel emulator. Once the receiver determines its port and hostname, it will write them to the "recvInfo" file as specified for the channel emulator.

GbnReceiver

This class provides an interface for the "main" to bind to a port number and receive data packets and send acknowledgement packets. It provides one public interface function called recvFrom(...), when invoked will open a socket and wait for data packets from the channel emulator. The following are the steps it follows:

1. recvfrom(..) the channel emulator
2. if an EOT packet is received, accepts it and terminates the class and destroys the object
3. if the packet's sequence number is the expected sequence number, then write the packet's payload to the file and acknowledge received packet, update the expected sequence number to the next valid number and continue
4. if the packet's sequence number is not the expected sequence number, display an "unexpected" error to the standard error output, acknowledge the packet saying that it has received that packet and continue

GbnSendr

This class also provides a similar interface for the “main” to execute. It provides a public interface function called sendTo(), whose functionality is to send and receive packets from a socket that is established after retrieving the channel emulator details from “channelInfo” file. GbnSendr class multiple private class functions which assist in performing the go-back-n protocol correctly. Here are the private class functions used by the class to send and receive packets from the channel emulator.

Purpose: To set the timeout for the entire window in order to resend packets accordingly

- setTimeOut(…)

Purpose: To receive acknowledgement packets, if the recvfrom(...) c-library call returns -1, it means that a timeout has occurred and the packets in the window have to be resent to the channel. Return a true value, stating that a timeout has occured. Else receive acknowledgement packets to check to see if the received packet exists in the current window. If there is a packet with the same sequence number as of the packet received, then acknowledge the packet. Assume a **cummulative acknowledgement** has occured and move the base of the window until the next expected sequence number (i.e. the next in line sequence number from the current received packet).

- listenForAck()

Purpose: To search for the given sequence number in the current window, and return a boolean value which states whether the given sequence number exists in the window or not.

- searchForPkt()

Purpose: To resend all the packets in the window to the receiver

- resendPkts()

The sendTo() public class function follows the following steps to receive packets from the channel emulator:

1. Open a socket
2. Validate the provided transfer file to read data from
3. if the current window already contains N(10) packets, then set the timeout and wait call the private listenForAck() function
4. Once the function returns, reset the timeout of the socket, and check to see if it actually timed out. If it has timed out, resendPkts() is called
5. If the current window has spots for more packets, then create a Packet object and add that into our current window and send it to the channel
6. Once all the packets are sent to the channel emulator, wait until all the packets have been acknowledged
7. Upon determining that there are no outstanding packets waiting to be acknowledged, send a EOT packet and wait for acknowledgement and terminate

Selective Repeat

The design of the selective repeat protocol is also divided across two classes called SRRecvr and SRSendr, where the SRRecvr class is implemented to perform as the receiver for the selective repeat protocol while SRSendr class is implemented to perform as the sender for the selective repeat protocol. This protocol follows the same exact steps as the go-back-N protocol for socket establishment for both the sender and receiver with the channel emulator.

SRRecvr

This class is relatively similar to most of the processes involved as that of GbnReceiver class. It provides a single public class function recvFrom() interface for the “main” to and send and receive packets to the channel emulator. That is where the similarity ends, this class handles the packets differently as it has to accommodate for multiple cases. Firstly, each of the packet has its own unique timestamp to determine whether it has to be resent due to timeout. Second, it has to buffer out of order packets upon receiving from the channel emulator. This class multiple private class functions to assist in satisfying all these requirements accordingly for the Selective Repeat protocol.

Purpose: To check the window buffer to see if there are any other packets that could be written to the file given the expected sequence number. If there are any other packets, they are written to the file and the base is moved to a the next expected sequence

- checkBufferWin(...)
Purpose: Given a sequence number, it determines if that particular sequence numbers' packet exists in the window buffer. Returns true or false accordingly
- inWinBuffer(...)
Purpose: To determine whether the currently received data packet is "fresh" in the sense whether it hasn't been written to the file yet or not. It will return true, if its a new packet with data that hasn't been written to the file yet, else false
- isFreshPkt(...)
Purpose: To determine whether the currently received data packet is within the range of the window buffer in order to either buffer it if its new or resend an acknowledgement packet as its already been buffered
- isValidPkt(...)
The recvFrom() public class function follows the following steps in order to send and receive packets from the channel emulator:

1. recvfrom(...) the channel emulator
2. if an EOT packet is received, check whether the window buffer is empty, if it is throw an error and continue. Else accept the packet, resend an EOT and terminate
3. If the received packet's sequence is the expected sequence, then write it to the file and reset the expected sequence and send an acknowledgement
4. else, validate the received packet to determine if its out of range, if its a valid packet, then check the buffer to see if it already exists or not, and whether it has already been written. If both the conditions are satisfied, add it to the window buffer
5. acknowledge the valid packet accordingly
6. if the packet is invalid, display an error and continue
7. Check the buffer window to see whether any of the packets could be written to the file based on the expected sequence number

SRSendr

The works under the same principle as that of the go-back-n variation of the sender. However, it has other things it needs to consider as, each individual packet will have its own timeout along with packets being acknowledged out of order. These requirements are satisfied by using the following private class functions.
Purpose: To set the timeout for specific packets in order to resend that particular packet upon timeout
- setTimeOut(…)
Purpose: To receive acknowledgement packets, if the recvfrom(...) c-library call returns -1, it means that a timeout has occurred and the nearest sequence number to the timeout will be to be resent to the channel. Return a the index of the sequence number in the window that has to be resent. Else receive acknowledgement packets. Set the acknowledged packet to true upon finding its index in the window using findSeqInWin(). If there is a packet with the same sequence number as of the packet received, then acknowledge the packet. Check the window for the next expected packet that has also been acknowledged and change the base of the window accordingly. Reset the timeout value to the packet that has been waiting for the longest time in the window.
- listenForAck()
Purpose: to find the sequence number provided in the window and return its location in relation to the index in the window
- findSeqInWin()
Purpose: To resend the packet at the provided window to the channel emulator for an acknowledgement from the receiver. Resend the packet and set the timeout to the next packet waiting in the window for the longest time to be acknowledged
- resendPkt(...)
The sendTo() public class function follows the following steps to receive packets from the channel emulator:

1. Open a socket
2. Validate the provided transfer file to read data from
3. if the window is full containing N(10) packets, then wait for a spot, by listening for acknowledgements from the channel

emulator

4. if the Timedout flag as been set, then the index returned by the listenForAck() function will be resent by calling the resendPkt()
5. if there are no items left to be acknowledged in the window, reset the timeout to the original timeout time and continue
6. Once a spot opens up in the window, send the packet, add it to the window, and get the current timestamp for the packet so that, given time it could be used to set the timeout for the packet
7. If the first packet flag hasn't been set, then set the initial timeout and continue
8. upon sending all the data to the channel emulator, wait for any outstanding packets that need to be acknowledged
9. Send an EOT packet and wait for the acknowledgement of the EOT packet. Once received terminate.

Testing

Testing for the protocols was done on various inputs that play a role in determining how each of the protocols react to the conditions of the sender; its timeout, and the channel emulator; with its probability and network delay. Multiple test cases were tested on each of the implementations.

Boundary cases:	File length:	Various data types:
- short timeout, long network delay, low loss probability	999 bytes = 2 packets	integers
- long timeout, short network delay, high loss probability	60000 bytes = 2 sets of 32 packets	strings
- short timeout, long network delay, high loss probability	45000 bytes = 1 and half set of 32 packets	integers and strings