CS118 Project 2: Reliable Data Transfer Protocol

Chris Konstad
504275045

Ty Giacalone
404001782

1 Implementation Description

1.1 Protocol overview

Our protocol has a few main message types: TRN (transfer), ACK (acknowledge), FIN (finalize), and FINACK (final acknowledge). Message types are distinguished in the packet header using a bit field.

In general, TRN packets are acknowledged using ACK packets with the same sequence number, and FIN packets are acknowledged using FINACK packets, which are packets with both the FIN and ACK flags set. The protocol is a bit more complicated than that due to the send and receive windows and selective repeat, but that is basically how the protocol works.

If a message is either lost or corrupted, the sender does not receive an ACK for that TRN packet and resends it after the timeout.

Packets are only sent a finite number of times before communication is assumed to be lost, in which case the transfer closes.

1.2 Packet layout

The packet header consists of 5 bytes of data. Since our maximum packet length was 5 thousand bytes, that leaves up to 995 bytes for holding data. Packets with a length of 0 in Figure 1 don't have any data bytes.

The maximum sequence number is 30000. The use of a receiving window helps calculate the total offset of a packet's data upon reception.

TYPE	DATA	LENGTH	SEQ	f_{FIN}	f_{ACK}
ACK	NA	0	seq	0	1
FIN	NA	0	NA	1	0
FINACK	NA	0	NA	1	1
TRN	data	length	seq	0	0

Figure 1: Logical packet designs.

1 byte	4 bytes	up to 995 bytes
$f_{ACK}, f_{FIN}, 0, \ldots, 0$	seq	data

Figure 2: Physical packet designs.

1.3 Timeouts

There is a user definable timeout before packets are considered lost. Since the receiver simply ignores corrupted packets, a full timeout period is required before resending corrupted packets. This design was chosen for its simplicity. When the sender does not receive an ACK packet for a TRN packet that it sent before timing out, it will resend the TRN packet.

Corrupted ACK packets are ignored, meaning the sender will resend the TRN packet until it receives a corresponding ACK.

1.4 Window and selective repeat

A window is specified for both the client and the server.

1.4.1 Sending with a window

1. The sender will send all of the packets that are not already acknowledged that are in the window.
2. It will then collect all of the ACKs that the receiver responds with, marking off each TRN packet that is acknowledged.
3. This collection ends with the first timeout of the packet receiving function.
4. Then, it will shift up the window to the first unacknowledged packet, and loop back to sending all packets in the (possibly new) window that are not acknowledged.

Please note that this technique only resends packets that an ACK has not yet been received for.

1.4.2 Receiving with a window

1. The receiver will calculate an offset using the packet's sequence number and the maximum sequence number allowed.
2. This offset is adjusted if the sequence number rolls over or if the window straddles the rollover point.
3. The window is shifted up if the offset if greater than the window's maximum position.
4. The buffer is grown if need be.
5. The data is copied into the buffer.
6. An ACK is sent using the sequence number of the TRN packet.

1.4.3 Finalizing the Connection

The sender sends a FIN packet when it is out of TRN packets to send. When the receiver gets the FIN packet, it sends a FINACK. The sender only attempts to send the FIN packet a finite number of times before assuming the connection is lost or closed. The receiver only sends the FINACK once before closing the connection.

2 Difficulties

One bug that took awhile to fix was when the server would switch from receiving to listening, it would receive the last FIN packets sent from the client that was trying to close the old connection. To fix this, the receiving code added a loop at the beginning that consumes any FIN packets before listening for TRN packets. Similarly, the sending code, if it receives a TRN packet instead of an ACK packet, will stop sending, letting the client quickly transition from sending to receiving.

Another bug that took some time to fix was how to handle closing a connection. This was fixed by limiting the number of sending and receiving time outs allowed before a connection is considered closed. The problem with this is that a technically working connection may be closed when it only has a really high packet loss rate, but the limits are high enough that probabilistically speaking multiple packets will get through before that happens. Plus, receiving any packet, corrupted or not, resets the counters.

The last hard bug to fix was calculating the total offset into the file a received packet's sequence number meant. When the window straddled the rollover point in the sequence numbers, the number of rollovers would be incorrectly tallied, causing a packet with a sequence number before the rollover point to be treated as a packet with an offset of its real offset plus another $CWnd$ length offset. This was fixed by adjusting the calculated offset if it was greater than $window.max + CWnd$.

Notes