

Fishnet Assignment 2: Reliable Transport

Due: May 30, 2002.

In this assignment, you will work in teams of one to four (you can use the same groups you had in the last assignment, or create a new group) to develop a Fishnet node that reliably transfers files to other nodes. The program you write builds on your solution to the first project. The goal of this assignment is for you to understand reliable transport.

1. What You Need To Write

Write a C program in a single file called hw2.c that does the following:

- Builds on top of functionality from the first assignment:
 - Takes three command line arguments as before, joins the Fishnet, performs the tasks below in any order and run until you enter ^D (end of file) or type exit, when libfish.a will end the program.
 - Accepts keyboard input commands of the form "send <nnn> <message>" and implements the Fishnet echo protocol, which is useful for testing. You should comment out all of the print lines from before except for "Recv" line (which makes sending useful!) to avoid clutter.
 - Maintains an up-to-date routing table via the distance vector protocol and forwards packets to remote nodes using the routing table. Again, you should comment out all of the output lines from before to avoid clutter.
- Waits to get a line of input from the keyboard of the form "sendfile <nnn> <filename>", then reliably transfers the contents of the file across the Fishnet to the destination node where it is written to a file. To stress reliability mechanisms, the Fishnet in which you run will drop a significant proportion of packets. Successful transfer requires that you implement three pieces of functionality, each of which are described below: a file simple transfer application that passes control information such as filenames; reliability using timers and retransmissions; and connection setup and teardown. You should use transport packets as defined in fish.h for all of these tasks.
- *Connections*. A connection is identified by the four-tuple of source and destination address plus source and destination port. To implement connection setup, the sender should send a packet with the SYN flag turned on and wait for it to be acknowledged; other packets in the connection should have this flag turned off. The receiver, on receiving a packet with the SYN flag on for a destination port for which there is an application should establish the connection state and

acknowledge the packet. (You do not need to handle concurrent connections. This means that you may not have sufficient resources to establish a new connection, in which case the sender will time out.) To implement connection teardown, the sender should send a packet with the FIN flag turned on and wait for it to be acknowledged and then clear connection state. The receiver, on receiving a packet with the FIN flag on and the expected sequence number for a connection that is established should notify the application that the connection has been closed and acknowledge the packet. It should not clear out the connection state just yet; see reliability below.

- *Reliability.* Reliable transmission should be achieved for each packet, including those with SYN or FIN set, by acknowledgement and retransmission. Each packet that is not a retransmission should have its sequence number incremented. The setup protocol above allows any initial sequence number, e.g., one. Receivers should acknowledge every packet that is part of a connection or establishes a new connection. The acknowledgement number should be that carried in the packet being acknowledged if that packet carries the sequence number expected by the receiver, otherwise the acknowledgment should be a repeat of the last acknowledgement. At the sender, a retransmission timer of RETRANSMIT_TIMEOUT seconds should be kept for each unacknowledged packet, and the packet should be re-sent verbatim if the timer fires. If MAX_RETRANSMIT copies of a packet have been unsuccessfully retransmitted then the connection should be aborted at the sender. At the receiver, if a connection has seen no packet activity for IDLE_TIMEOUT seconds, then the connection state should be deleted.
- *File Transfer.* The file transfer application should read from the local send file named in the command, transfer its contents over the network, and write them to a local receive file. The first packet of your file transfer, for which the SYN bit is set, should carry only the null-terminated name of the file being transferred. The name of the receive file should be the source node address of the connection, followed by a dash, followed by the name of the file being transferred, e.g., "23-myfile.txt". Note that file transfer can be a significant security vulnerability, and you should take the following steps to avoid compromises: do not accept filenames containing the character "/"; make sure the permissions of the written file do not allow it to be executed (and do not source or otherwise interpret it yourself); and write the file truncating any existing copy. If you cannot open the file for writing, then you should consider the connection to be closed at the receiver (and the sender will soon timeout and abort its connection). The remaining packets should carry the contents of the file in order. The output file is closed when the connection is torn down. The final FIN packet should not include other data. (These rules about SYN and FIN packets carrying data are arbitrary, chosen only because we have a simple implementation in mind.) File transfer packets should be sent with destination port 20 (FTP Data) and a source port that is determined by the sender.

- Your program should print the following output during file transfer:
 - At the receiver when a transfer is started: `"Receive <filename>\n"`; at the sender you will already see the `"sendfile"` command. Then at the sender and receiver when a file transfer has successfully completed: `"\n<xxx> bytes\n"`, showing how much data was transferred. At the sender, you should not print this message until the receiver has acknowledged the FIN packet. Then, you will know that the file has been successfully transferred when you see this message and the number of bytes matches the length of the file. If the connection instead times out then you should print `"\nTransfer timed out\n"`.
 - At the sender and receiver the following single letter codes, without a newline, when a packet of the appropriate type is sent or received:
 - "S" for a SYN packet
 - "F" for a FIN packet
 - "." for a regular data packet
 - ":" for an acknowledgement packet
 - "!" for a retransmission at the sender or duplicate at the receiver
 - "?" for an out-of-order packet at the receiver
 - "x" when connection state is deleted at the receiver

You should read about and call `flush()` after printing single letter codes so that they appear on the console without delay. If you print the codes as specified above, a successful connection will appear as a sequence of mostly dot characters marching over your screen.

2. Step-by-Step Development and Test Instructions

Here is a suggested set of steps to develop the required functionality.

1. Start with `hw1.c` by copying it to the new file `hw2.c`. If your homework #1 did not work, we will be providing a reference implementation for you to use. Check the `~cs123b` directory for this.
2. Run the fishhead with high loss (try `"--help"` or just `"—loss 0.2"`) to stress the reliability mechanisms.
3. Code the `"sendfile"` command syntax, but just send a series of dummy text packets reliably (letting the first received packet start the connection and handling acknowledgement and retransmission as necessary) and print them out to the console at the receiver.
4. Add connection setup and teardown code at both sender and receiver, plus the print messages, and test a dummy transfer.

5. Add the file transfer code by letting the first packet carry the filename, reading from the input file at the sender and writing to the output file at the receiver, keeping track of bytes transferred and printing the remaining messages.
6. We will be providing a class reference solution (again in ~cs123b). Test your program for interoperability with this implementation in preparation for joining the class Fishnet.
7. You're done! Read and do any turnin work now.
8. *For Fun*. Implement the ability to handle multiple concurrent connections by replicating the connection state.
9. *For Fun*. The acknowledgement rules are defined in such a way that your sender can support a larger sliding window of 8 packets (say) without any change in the receiver code. So make your sender use a larger sliding window and see how much faster it is.

3. Turn In and Discussion Questions

This time, you turn in your program source code by transferring it over the Fishnet to our node 1. You hand in a paper copy of the discussion questions and test cases below as well as your source code.

1. Turn in your program by transferring it to our node running on the class Fishnet. We will send details including what name to use on the class mailing list. (2 points for successful transfer.)
2. The above turnin is the test case. Save the output and print it for us, along with a paper copy of your program. (12 points for the program itself and demonstration of how it works via the test case output.)
3. Question (2 points): What difference (advantage or disadvantage in terms of reliability) does having a SYN flag make compared to simply using a sequence number of one to signify the start of a new connection?
4. Question (2 points): What difference (advantage or disadvantage in terms of reliability) does having a FIN flag make compared to simply not signaling FIN and letting old state be deleted as it is already?
5. Question (2 points): In what ways does the system you have built fall short of perfect reliable file transfer? You should assume that the data in packets is protected by a 32 bit checksum while in transit (and this is in fact the case).

Note: There are specific, short answers to the above questions. Don't write much!