Fishnet Assignment 3: Reliable Transport

Due: Friday Feb 21, 2003, at the beginning of class. Out: Monday Feb 3, 2003.
CSE/EE461 Winter 2003; Wetherall.

The purpose of this assignment is for you to understand and implement a protocol for reliable transport, your own simplified version of TCP. You develop this transport protocol within your node, along with a command to transfer data that you can use to test it. The rest of your node implements the routing, flooding and naming functionality that you have already built up. In a following assignment, you will develop separate applications that use your Fishnet node.

1 Reliable Transport

Your job is to design and implement a transport protocol with the following features:

- *Reliability*. Each packet should be transmitted reliably by using sequence number and acknowledgement number fields and timeouts and retransmissions. The sequence number advances in terms of bytes of data that are sent. The acknowledgement number operates as in TCP to give the next expected in-order sequence number, and an acknowledgement should be sent every time that a data packet is received. That is, the acknowledgement number does not increase when a packet has been lost until that packet has been retransmitted and received. You can use constants in fish.h for the value of the timeout and the maximum number of times to try retransmitting before giving up. You should implement a "stop and wait" style scheme only, where a single packet can be outstanding at a time. A better implementation would use a fixed-size sliding window, but this is a considerable step up in complexity. An excellent implementation would use a dynamically-sized sliding window depending on acknowledgements and loss, mimicking the operation of TCP. We consider the latter challenging – please let us know if you are able to achieve this.

- *Connection state machine*. A connection is identified by a four-tuple, the combination of a source and destination fishnet address plus a source and destination port value. Read Peterson 5.3.2 before you implement your own connection state machine. Each connection should be established before data is transferred, and torn down after all data has been transferred. The SYN flag on a packet sent from A to B is used by A to request that the half of the connection from A to B be established. That half is established when B acknowledges the SYN packet sent by A. The half of the connection in the reverse direction is established similarly. The FIN flag on a packet sent from A to B is used by A to request that the half of the connection from A to B be torn down. That half is torn down when B acknowledges the FIN packet sent by A. The half of the connection in the reverse direction is torn down similarly. Finally, the RST flag on a packet sent from A to B is used to abruptly abort the connection in case of error. It should also be used to indicate "connection refused" when there is no application awaiting connections on the destination port. You must support multiple, concurrent connections. We suggest that you define your own transport connection structure, with your node having an array of such structures, one per connection in use. The structure will encode all state associated with a connection, including sequence numbers, buffered data (both sent awaiting acknowledgment and possible retransmission, and received awaiting processing by the application), and connection state (such as established, the SYN has been sent but not acknowledged, etc.).

- *Flow control*. Your protocol should use the advertised window field along with the sequence and acknowledgement numbers to implement flow control so that a fast sender will not overwhelm a slow receiver. The advertised window field tells the other end how much buffer space is available to hold data that has not yet been consumed by the application. Note that this will not be much of an issue in this assignment (as the in-node protocol may process data immediately, as it arrives, rather than hav-

ing to buffer it until a separate application is ready to receive it) but it must work and will be needed by the next assignment.

You must use only packets of type `struct transport_packet` to build this protocol. As usual, you should strive to come up with a design that will interoperate with other students' nodes. As you do, take the following steps:

1. Build a "transfer" command into your node that, on the sender side, sends sets up a connection to another node and sends a well-known test pattern to the other side, and tears down the connection. On the receiver side, your node should check that the test pattern is expected and provide feedback about the success or failure of the overall transfer. This command is purely an expedient way to test your transport protocol. For the transfer pattern you should use a configurable number of fixed sized packets, 512 bytes by default, whose contents are all bytes with values of N for the Nth packet, e.g., all 1s for the first packet, 2s for the second, etc., and using a specific port, 1, for both source and destination.

2. At the sender and receiver, print 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 a retransmission at the sender or duplicate at the receiver
   - "`:`" for an acknowledgement packet that advances the acknowledgement field
   - "`?`" for an acknowledgement packet that does not advance the field

   These codes will give you visual feedback to help you gauge the progress of a transfer and give us a trace for your turnin. 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.

3. Run your Fishnet with a relatively high level of packet loss (5%, say) to check that lost data is successfully retransmitted. Packet loss is a command-line option to fishhead.

2 Discussion Questions

a) Your transport protocol implementation picks an initial sequence number when establishing a new connection. This might be 1, or it could be a random value. Which is better, and why?

b) Your transport protocol implementation picks the size of a buffer for received data that is used as part of flow control. How large should this buffer be, and why?

3 Turn In

Turn-in all of your source code and Makefile for your entire node electronically. It should make using the target hw3. Bring a printed copy of your transport protocol code to class (you don't need to include the routing and naming code from earlier assignments). Bring a printout of a 100 packet reliable transfer running using your nodes on a two node private fishnet, plus answers to the discussion questions.

—END—