Fishnet Assignment 4: Applications

Due: Friday, December 3 at the beginning of class
CSE/EE 461, Autumn 2004

The purpose of this assignment is for you to understand the structure of network applications by developing two applications (Fishfone and Fishster) that communicate over the network by using the fishnet code you have already developed. Because each involves a server, we define the following “well-known ports”: the fishfone server listens on port 20 and the fishster server listens on port 80. Clients of these services are assigned an unused port (to wait for replies).

Because multiple services and multiple applications must co-exist in your code, pay particular attention to structuring your code to make it easy to add additional services and clients in a modular fashion.

Note that we expect building the application interface and making your transport protocol sufficiently robust for these applications to be a large part of the work for this assignment.

You have a couple of design options:

- You may build event-driven applications that plug into your transport protocol.
- Or, you may build a synchronous interface for applications written as separate Ruby programs. You may use Amphibian, which is distributed on the course web site, to help you with this.

1 Fishfone

The first part of this assignment is to design and implement a Fishfone application that provides a walkie-talkie service between IPAQs with the following features:

- *Transport Usage*. Your application should be able to both (i) connect to a remote application to send audio samples over the network, and (ii) accept connections from a remote application to receive audio samples over the network. To do this you should use the transport protocol you developed in the previous assignment. Your Fishfone should accept connections on the Transport port reserved for Fishfone, port 20. Each connection should be used to send data in one direction, from the client initiating the call to the server accepting the call; thus you need to use two one-way connections to have a two-way conversation.
- *Half-Duplex Audio*. Unfortunately, your IPAQ does not allow you to simultaneously play and record sound. To work around this, your application should support both operations, but only do one of them at a time. We suggest you detect whether a button is depressed on the IPAQ and use this as a user interface control for switching between send and receive mode. That is, when a button is depressed, you should establish a connection, take audio samples from the microphone on your IPAQ and continue sending them while the button remains depressed, and stop taking audio samples and close the connection when the button is released. Otherwise, when the button is not depressed, you should accept connections from other Fishfones and receive samples from over the network, and send them to the speaker on your IPAQ while the connection remains open.

The program soundtest.rb, available on the course web under the “Using sound on the iPAQ” link, illustrates how to use the microphone and speaker. The program waits for the user to press the upper left button on the side of the IPAQ, then records sound from the microphone for as long as the button is depressed, and then plays the recorded sounds back when the button is released. Note that the playback is softer than the ambient sound; you need to speak clearly into the IPAQ to hear anything played back. Note that, unlike previous assignments, you cannot run and test your code in a simulated or emulated Fishnet because the development environment does not have the same microphone and speaker devices as the IPAQs.

When you are finished, you should be able to call up your classmates on your Fishfone and talk to them (e.g., even when the data must be relayed across multiple IPAQs to reach the destination in the next room).

2 Fishster

The second part of the assignment is to design and implement Fishster, a way of sharing files with your classmates. As file sharing was the original motivation behind the Web, we build Fishster using web protocols as a key building block. You will need to build:

- *Simple Web server*. Your server should accept connections on the Transport port reserved for Fishster, port 80. For each arriving connection, it decodes the request using HTTP (see Peterson, Chapter 9.2.2).
    - Note that standard HTTP is implemented on TCP, which uses bidirectional connections, while our Fishnet transport protocol provides only unidirectional connections. To account, for this, we will add a new request header of the form “Response-Port: 7”. The client uses this header to indicate the port it will use to listen for responses (e.g., port 7). Similarly, HTTP responses should echo the name of the requested file in a header such as “Location: index”.
    - You should only support the very simplest form of “GET” requests, as in “GET index http/1.0<crlf>Response-Port: 7”. The server then establishes another connection in the opposite direction, and if the file exists in the IPAQ’s “/www” directory, sends the file to the other side, again using HTTP. Use only the simplest reply format, e.g., “HTTP/1.0 202 Accepted<crlf>Location: index<crlf><file contents>", or “HTTP/1.0 404 Not Found<crlf>Location: index<crlf>” if the file doesn’t exist.
- *Fishster client.* On the client side, the application should be able to connect to a remote web server running on an IPAQ at port 80. Instead of displaying the contents of the file (as in a normal web browser), your Fishster client downloads all content that it doesn’t already have. The file “index”, if it exists, contains a list of file names that the server is willing to share. Note that we are using HTTP/1.0, so that connections are terminated as soon as the file has been downloaded; another connection must be established to download additional files. To avoid having every file downloaded over every hop multiple times, the Fishster client should only connect to web servers that are within one hop of the client; provided the network stays up, all unique files will eventually end up on all nodes. (This means that as you find and download files, you need to add them to your “index” file.)

As a test case, set up the file “index” to point to a file named after your CSE login, and create a file of that name with contents of your choice (e.g., a random #).

3 Discussion Questions

1. Did you use an event-driven or a synchronous architecture for your applications? Give one advantage and one disadvantage of the approach you chose.
2. The Fishfone as described above uses many transport connections. A simpler design would be to use only a single long-lived connection and send data across it in both directions. Describe the pros and cons of this design compared to the one above.
3. Describe which features of your transport protocol are a good fit to the Fishfone application and which are not. Are the features that are not a good fit simply unnecessary, or are they problematic, and why? If problematic, how can we best deal with them?
4. Describe which features of your transport protocol are a good fit to the Fishster application and which are not. Are the features that are not a good fit simply unnecessary, or are they problematic, and why? If problematic, how can we best deal with them?
5. Describe one way in which you would like to improve your design.

4 Turn In

We will set up timeslots for each group to demonstrate their code during the last week before finals (details TBA), and we will collect your iPAQs at these meetings.

1. Electronically turn in all of the source code for your entire node.
2. In class on the due date, turn in your design document, a printout of your code, and answers to the discussion questions.
3. During your scheduled timeslot, provide us with a live demonstration of your Fishfone running on your iPAQ communicating with our iPAQ.
4. Use Fishster to collect as many files as you can from members of the class. Also record which node you received each file from. Email a list of the names of files you received, as well as a list of files that you provided to Fishster, to ivmaykov@cs by 5pm on December 6.