Fishnet Assignment 4: Applications

Due: Tuesday, December 9, 2003 at 5pm
CSE/EE 461: Fall 2003

The purpose of this assignment is for you to understand the structure of network applications by developing three applications (fishfone, fishchat, 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; the fishchat server listens on port 40; 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.

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; you need to use multiple 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 signal 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 lasts.

The attached file, soundtest.rb, illustrates how to use the microphone and speaker. The file waits for the user to press the upper left button on the side of the IPAQ, then records off 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 private 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 Fishchat

The second part of this assignment is to design and implement a Fishchat service with the following features:

- *Fishchat server*. Your fishchat server should accept connections on the Transport port reserved for Fishchat: port 40. For each arriving connection, it establishes another connection in the opposite direction. Fishchat is a multiway text conferencing system, and so the server must be able to accept and manage multiple connections simultaneously. Once two or more sources are connected (e.g., including the IPAQ hosting the service), all bytes sent to the server are "relayed" to all other IPAQ's listening to the service. The server should separate text by source by sending it line by line, labeled each line with the name of the client who sourced that particular message.

- *Fishchat client.* On the client side, the application should be able to connect to a remote server (e.g., using the name of the IPAQ) at the Fishcat port, and then display all text relayed back from the server. Since you'll want to be able to use your own IPAQ as both a client and a server, you should design some way for your IPAQ to connect to itself (of course, the bytes in this case are not sent over the network, but relayed internally).

3 Fishster

The third 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). You should only support the very simplest form of "GET" requests, as in "GET index http/1.0<crlf> Host: ipaq-tom<crlf>". 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><crlf><file contents>", or "HTTP/1.0 404 Not Found<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 protocol to follow is that 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 termi-

nated 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 your CSE login, and create a file of that name with contents of your choice (e.g., a random #).

4 Discussion Questions

1. 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.

2. 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?

3. How did you structure your code to be able to manage the multiple simultaneous connections required for fishchat? How would your design differ if application code (clients and servers) were in a separate process from your fishnet code, communicating via socket calls?

4. Describe one way in which you would like to improve your design.

5 Turn In

NOTE: even with slip days, all aspects of this assignment must be turned in by 5pm, Friday, December 12. We will set up timeslots for each group to demonstrate their code on Dec. 11 and 12, and we will collect your IPAQ's at these meetings.

1. Turn-in all of your source code for your entire node electronically.

2. Provide us with a live demonstration of your fishfone and fishchat running on your IPAQ communicating with our IPAQ.

3. Use fishster to collect as many name files as you can for the various people in the class. Also record which node you received each file from. Email your list, along with the name file you sourced, to harsha@cs by 5pm on Dec. 12.