Introduction to Fishnet

CSE/EE461 Autumn 2002; Wetherall.

This informational handout covers introductory material describing the Fishnet software that you use, the Fishnet programs that you write, the development environment, and testing strategies.

1 What is Fishnet?

Fishnet is a class-sized network that we will build over the quarter! You will each build one node of the network, working in pairs, over a series of assignments. Once we are done, nodes from each team will be able to route messages and share files over a collaborative wireless network that runs on the IPAQs.

The term "Fishnet" is somewhat overloaded, and before going further we want to clarify what it means. First, it is the code that makes up all of the assignments. This is the *Fishnet development environment*. Second, it is a running network of many nodes. This is what we most commonly mean by *Fishnet*. Note, however, that there are many potential running Fishnet networks. You will all start your own, private, fishnet as a managed overlay controlled by a single fishhead (a key component to be described shortly) to develop and test the code required for assignments. It will run as cluster of processes on one or more PCs. In this form your nodes will not interact with nodes of other students' networks. There is also The Fishnet, with a capital "T". This is the one, shared network that everyone's node is able to join and participate in a collective class network. It is run as a wireless 802.11 ad-hoc network with one node per IPAQ.

1.1 Fishnet Components

The Fishnet project code, like everything else you need, is available from the course web site. The package we provide you with contains the following key components. There are also different subdirectories for the x86 and ARM platforms.

1.1.1 fishhead

fishhead is a program that manages fishnet nodes as a managed overlay for your development and testing on your own private fishnet; we use the IPAQs without a fishhead for the production, class Fishnet. A fishnet can contain many nodes, but only one fishhead. The main function of the fishhead is to tell individual nodes who their neighbors are. It is important to understand that the fishhead manages the network topology and decides who is connected to whom, not you in your programs. Similarly, radio range determines the network topology in the wireless, ad-hoc network formed by the IPAQs, not you in your programs.

As you develop your Fishnet node and test it, one of the first things it will do is to join a private fishnet network by contacting the fishhead. This means that before you start one of your Fishnet nodes there must be a fishhead process running. You only need to start a fishhead for your network once, even though the nodes in the network can come and go. Type "`fishhead --help`" to get usage information.

1.1.2 libfish.a

The fish library implements all of the Fishnet functionality that you will need for your assignments. When you send a message using the `fish_sendframe()` function, for example, libfish.a is called to do the work of sending the packet. libfish.a also prints a large (but controllable) amount of debugging information to the console (stderr) to help you understand what is going on with your program. The library source code is available in the fishsrc directory so that you can see how the Fishnet really works and to help with your debugging, but you MUST NOT change this code at all so that you remain compatible with other people's nodes.

1.1.3 fish.h

This is the header file that you should include in your C program to gain access to the functionality implemented in the fish library. It contains the dozen or so Fishnet API functions that you can call, as well as the structures that define packet formats, and other Fishnet constants. You should read the comments in this file, as it contains the definitive and only Fishnet API documentation.

2 Hello World in Fishnet

The first exercise you should complete is to learn how to use Fishnet is to compile, run, and inspect the "hello world" program that is bundled with the distribution. This does not involve the IPAQs.

1. Check that you can get into the CSE Lab and log into a Windows machine. From there, check that you can remotely access one of the Linux servers (`fiji, tahiti, sumatra, ceylon, blackbox02, blackbox03, blackbox04, blackbox05`). If you are not a CSE major, you must fill out a request form in order to gain access and accounts; the form will be available in class and at the CSE front desk. All of your work will take place on the Linux servers. The easiest way to work is to use teraterm to ssh to these machines; there is a shortcut to do this on most desktops. You can also use any X windows packages that you are familiar with. Alternatively, you can use a Linux PC of your choice, but note that we have not tested Fishnet for environments other than the CSE Lab and the IPAQs.
2. Download the Fishnet package from the course web pages into your home directory and unpack it. The easiest way to do this is to run the lynx browser (`lynx http://www.washington.edu/461`), follow the Fishnet links, and download the tarball. Alternatively you could use `wget` to download the tarball. To unpack the tarball, use `tar` with the arguments given on the web page. You should now have a fishnet directory with files in it. As a general note, you can use the `man` command to get a help page for any Unix command or function we ask you to use that you don't understand.
3. Compile the program hello.c. to make a Fishnet node. There is a target to do this in the supplied Makefile, so you can simply type "`make hello`"

4. Set up your own private, fishnet using the "hello" node. Read the comment documentation in hello.c to do this. In a nutshell, first start a fishhead process to manage your own private fishnet by running the fishhead program that is inside the fishnet directory. Type "`./fishhead --help`" to find out what command line arguments it accepts. You will need to give it an argument to indicate which port it should listen on for messages. Pick a number between 1024 and 32K. If the number you choose is the same as someone else's, fishhead will fail to start. The fishhead program runs indefinitely to keep the Fishnet it is running up. When you're done, stop it by typing ^C. Second, create other windows and start hello nodes in each. Again, use "`--help`" to discover the command-line arguments it takes, and "`help`" when it is running to discover what commands the node understands. You can use these commands to send packets from one node to another.
5. Congratulations, you just ran a Fishnet!

After you have run the hello program, you should look at hello.c to understand what happened. Hello.c is the simplest Fishnet program – your nodes will be more complex. However, there are two aspects of hello.c that you will be replicating in your assignments.

Fishnet API is in fish.h. As you look at hello.c, you will see that it calls functions with names like `fish_send()`. These are Fishnet API calls. All of these calls are explained and documented with comments in the include file fish.h. There aren't that many. The documentation includes comments for the structures representing packets that are passed to and fro. You won't find this information anywhere else.

Fishnet Program Structure. All Fishnet programs have a similar structure. They are event-driven, like much real world code and especially any programs with GUIs that you may have written. What this means is that they typically use `main()` to initialize and then enter the "main event" loop. Initializing involves joining the fishnet with `fish_joinnetwork()` and setting up functions to be called on specific events, e.g., a `recv_input()` function to be called when there is keyboard input. Entering the main event loop is done by calling `fish_main()`, a call that never returns and from which the program is terminated directly when "`exit`" is typed at the keyboard. Events such as packet reception are handled as "upcalls" from the main loop.

3 Development and Testing

We will be developing under Linux and writing C code. Typically, you will go the CSE Labs, log into a Windows machine, and from there remotely access a Linux server using teraterm (ssh) or X windows. All programming, compiling and debugging of the Fishnet will take place on the Linux machines with Fishnet being used in managed overlay mode. In this mode, each node of your private fishnet will run as a process on a Linux machine. The compiler we will use is gcc, and the debugger we will use is gdb. You can use any editor you prefer, such as emacs.

When your program works, you will cross-compile it for the IPAQs, copy it over to your IPAQ using the building 802.11 wireless LAN, place your IPAQ in "ad-hoc" mode, and then run it to join the live class Fishnet.

Here are some other points you should note:

1. For your assignments, develop the code for your node in a file called hw1.c, hw2.c, etc. in the top-level directory created when you unpacked the Fishnet distribution. We have supplied a Makefile so that you can type "`make hw1`" etc. to compile your program. This will invoke the compiler, gcc, with the right command line arguments and build x86 executables. If you want to build ARM executables for the IPAQs then type "`make $ARM=1 hw1`" etc.
2. You access the Fishnet API by including the file fish.h in your program.
3. The fishsrc directory contains the source code that we have written so that you may see it and use it for debugging, but you MUST NOT change it.
4. You will find at least the following C library functions helpful: atoi, printf, sscanf, strlen, and strncpy. Remember than you can type "`man <functionname>`" to get help for using these functions.
5. Remember that only feasible way to develop software of any complexity is to find a way to break the task down into simpler, checkable, subtasks. You might first write a program that had a `main()` function that simply accepts the command line arguments, prints them out, and exits. When this compiles and runs properly (experience shows most of you will have found and fixed at least one error already!) move to the next task. The next task might be joining the network and sitting in `fish_main()`. You can use debugging messages to check if this program has worked, as you will get messages when you have joined. You can quit the program by typing the command "exit". Next, you might try for keyboard input, and print it out but not send a packet yet, and so forth. While this strategy might seem like overkill at this stage, you should get used to it before attempting the larger assignments, and particularly when you find that something is not working properly. The beauty of testing subtasks is that the problems are rarely where you think they are (otherwise you wouldn't have made the mistake) and testing is the only way to pin them down. Despite this strategy, more difficult bugs will require you to use the debugger, gdb.
6. The first way you should test your code when it is written is to run multiple copies of your node in your own private fishnet, as you did for "hello". You will need to use different command line arguments to select different node addresses. You will find it easiest to bring up a separate window for each node, as well as the window for the fishhead. Unless you've disabled them using the `fish_setdebuglevel()` function, you will see libfish debugging messages that show how your packets move through the network. Also, note that there are Fishnet API functions for logging debugging output to a file for later inspection.
7. The second way you should test your code is to run it with a combination of your node and our sample executable. This will check whether the different versions are interoperable. You MUST have an interoperable node before joining the live class network, or you may cause that network to experience problems.
8. You should write one program (per assignments) that runs in either the managed overlay on the CSE Lab clusters or on your IPAQ depending on its command-line arguments. (Hint: use fewer arguments for your IPAQ since it doesn't have a

keyboard to type on!) In both cases your node must join the Fishnet, but the arguments to the call are different. Also, if you are running on the IPAQ you must use only your assigned Fishnet address. It is the small number written on the bottom or side of your IPAQ. When running as part of your private fishnet you choose your own node addresses.

9. Instructions for dealing with the IPAQs can be found in the IPAQ FAQ on the class web page. Once you have a working, interoperable program you are ready to join the live network on the IPAQs. There are four steps. First, cross-compile your program for the IPAQ. Second, make sure your IPAQ has its 802.11 card in managed mode so that it is reachable from the CSE Lab machines via the building wireless network. Third, copy the program to your IPAQ using a file transfer utility. Fourth, to join the live class Fishnet you must switch the 802.11 card on your IPAQ to ad-hoc mode so that it can communicate with other IPAQs rather than with the rest of the Internet. Once this is done, simply run your node program and see what happens!

—END—