TCP for Minet

Project Part B

Overview

In this part of the project, you and your partner will build an implementation of TCP for the Minet TCP/IP stack. You will be provided with all parts of the stack except for the TCP module. You may implement the module in any way that you wish so long as it conforms to the Minet API, and to the reduced TCP standard described here. However, Minet provides a considerable amount of code, in the form of C++ classes, that you may use in your implementation. You may also earn extra credit by implementing additional parts of the TCP standard.

The Minet TCP/IP Stack

The Minet TCP/IP Stack is documented in a separate, eponymous handout. The low-level details of how Minet works, including the classes it makes available to you, the modules out of which the stack is constructed, and how the modules interface with each other is documented in that handout. Of course, it also doesn't hurt to look at the code. You will be given the source code to all of the Minet modules except for tcp_module and ip_module. You will find udp_module.cc quite helpful to begin. You will also receive binaries for ip_module, reader, and writer.

It is vital that you use the reader and writer binaries that we give you. These programs must be run as root or else they will not work. Because we don't want to give you root access, we are giving you binaries that are setuid to root. This means that these programs (and only these programs) run as root. If you would like to use reader and writer on a machine outside the TLAB, you will need root privileges on that machine.

Your IP Addresses

Each project group was assigned 255 IP addresses to use for the rest of the quarter. These address are of the form 10.10.x.y, where x will depend on your group id and y will range from 1 to 255. These addresses are special in that packets sent to them will not be forwarded beyond the local network. In fact, if you are using machines other than the TLAB machines, you will need to add a route so that they actually make it to the local network.

Dedicated TLAB Machines

You may use any of the TLAB machines, either from the console or remotely via ssh. In fact, you'll usually want to use two of them simultaneously. We have dedicated TLAB-11 through TLAB-15 to running Linux for the duration of the quarter. These machines should always be available for remote or console login. If they are not, send mail to request@cs.cmu.edu.

TCP Specification

The core specification for TCP is RFC 793, which you can and should fetch from www.ietf.org. In general, you will implement TCP as defined in that document, except for the parts listed below.

- You only need to implement Go-Back-N
- You do not have to support outstanding connections (i.e., an incoming connection queue to support the listen backlog) in a passive open.
- You do not have to implement congestion control.
- You do not have to implement support for the URG and PSH flags, the urgent pointer, or urgent (out-of-band) data.
- You do not have to support TCP options.
- You do not have to implement a keep-alive timer
- You do not have to implement the Nagle algorithm.
- You do not have to implement delayed acknowledgements.
- You do not have to generate or handle ICMP errors.
- You may assume that simultaneous opens and closes do not occur
- You may assume that sock_module only makes valid requests (that is, you do not have to worry about application errors)
- You may assume that exceptional conditions such as aborts do not occur.
- You should generate IP packets no larger than 576 bytes, and you should set your MSS (maximum [TCP] segment size) accordingly, to 536 bytes. Notice that this is the default MSS that TCP uses if there is no MSS option when a connection is negotiated.

Chapter 3 of your textbook also serves as an excellent introduction to TCP concepts and should be read before the RFC. **You will also find the TCP chapters in Rick Steven's book, "TCP/IP Illustrated, Volume1: The Protocols" extremely helpful. They will show you what a working stack behaves like, down to the bytes on the wire.**

Recommended Approach

There are many ways you can approach this project. The only requirements are that you meet the TCP specification detailed above, that your TCP module interfaces correctly to the rest of the Minet stack, and that your code builds and works on the TLAB machines. We recommend, however, that you use C++ and exploit the various classes and source code available in the Minet TCP/IP stack. Furthermore, we recommend you take the roughly the following approach.

1. Read Chapter three of your textbook
2. Skim RFC 793 and the Stevens chapters.
3. Read the "Minet TCP/IP Stack" handout.
4. Fetch, configure, and build Minet if you have not already done so. For this project, build Minet using "make project_b". It is important that you use /usr/local/bin/g++.

5. Examine the code in tcp_module.cc and udp_module.cc. The TCP module is simply a stub code that you need to flesh out. It just connects itself into the stack at the right place and runs a typical Minet event loop. UDP module is a bit more fleshed out. It has almost exactly the same interface to the IP multiplexor and to the Sock module as your TCP module will have.
6. Extend the TCP module so that it prints arriving packets and socket requests. You should be able to run the stack with this basic module, send traffic to it from another machine using netcat (nc), and see it arrive at your TCP module. You may find the classes in packet.h, tcp.h, ip.h, and sockint.h to be useful. Now is a good time to familiarize yourself with Minet's various environment variables. You should check out what the various MINET_DISPLAY options do.
7. Learn how to use MinetGetNextEvent's timeout feature. You will be using this to implement TCP's timers.
8. Write a class that represents the state of a connection. Think carefully about what this should contain. Think of a connection as being a finite state machine and consider using the states described in RFC 793. You may find the various classes in constate.h to be helpful here. In particular, your connection should have various timers associated with it. Your connection will also probably have input and output buffers associated with it.
9. Write a class that maps connection addresses (the Connection class) to connection state. Again, you may find constate.h to be helpful.
10. Add code to your TCP module to handle incoming IP packets. Begin by adding code to handle passive opens. Even without the Sock module, you can test this code by using a hard-coded connection representing a passive open. Note that the element of time enters in here. You will need to use one of your timers to deal with lost packets. You may find the classes in tcp.h and ip.h to be useful.
11. Add code to your TCP module to handle active opens. Again, you do not need to use the Sock module here. You can hard code the active open for now.
12. Add code to your TCP module to handle data transfer. Again, note the element of time and think of how to implement your timers using MinetGetNextEvent. Remember that you do not have to implement congestion control, only flow control. A good approach to data transfer is first to implement a Stop-And-Wait protocol and get it working, and then extend it to do Go-Back-N.
13. Add code to your TCP module to handle closes. Note that at this point the interface for passing up a remote close to socket is ill-defined. We will fix this as we go. Your code should be able to handle a remote close even if it doesn't push it up to the socket layer.
14. At this point, your TCP module should be able to carry on conversation with a hard-coded partner. Congratulations! You are finished with the most difficult part!
15. Re-read the discussion of the interface between the Sock module and the TCP module in the Minet TCP/IP Stack handout.
16. Make sure you understand the SockRequestResponse class. SockRequestResponses will advance your connections' state machines just like IP packets do. They will also affect the set of outstanding connections (item 9).

17. Add code that keeps track of outstanding requests that your TCP module has passed up to the Sock module. Recall that the interface is asynchronous. When you send a request to Sock module, the response may arrive at any time later. The only guarantee is that responses will arrive in the same order that requests were sent.
18. Add code to support the CONNECT request. This should simply create a connection address to state mapping and initiate an active open.
19. Add code to support the ACCEPT request. This should simply create a connection address (with an unbound remote side) to state mapping and initiate a passive open.
20. Add code to pass incoming connections on a passively open connection address (one for which you have received an ACCEPT) up to the Sock module as zero byte WRITE requests.
21. Add code to support the CLOSE request. This should shut down the connection gracefully, and then remove the connection
22. Add code to support the WRITE request. This should push data into the connection's output queue.
23. Add code to send new data up to the Sock module as WRITE requests. Note that the Sock module may refuse such a WRITE. In such cases, the TCP module should wait and try to resend the data later. We are currently working on a better flow control protocol between the Sock module and the TCP module.
24. Verify that you are generating and handling STATUS requests correctly.
25. Now try to use your stack with an application. tcp_server and tcp_client should work.
26. Now you ought to be able to use your http_client and http_server1 from the project A, simply by running it with "U" instead of "K". The more advanced http_servers will probably not work since the socket module's implementation of select is buggy. Note that while a Minet stack currently supports only a single application, you can run multiple Minet stacks on the same machine or on different TLAB machines to test your code.

Extra Credit: Congestion control

For extra credit, you may implement the congestion control parts of TCP as they are described in your textbook and in the other sources. Please note that while this is not much code, it does take considerable effort to get right.

Caveat Emptor

The Minet TCP/IP Stack is a work in progress. You can and will find bugs in it. There are three bugs that we are currently aware of

- The first IP packet sent to a new address is dropped. This is not actually a bug, but rather an implementation decision. What's going on is that ip_module will drop an outgoing packet if arp_module has no ARP entry for the destination IP address. However, arp_module will arp for the address and so the next packet sent to the destination address will probably find an ARP entry waiting for it. One way to "populate" arp_module so that you don't have to worry about this is to ping your stack from the destination IP address.

- TCP/Socket interface is incomplete. tcp_module can't report passive close to socket layer. This makes reusing connections difficult. The work-around is to rerun the stack on every connection. We will fix this.
- Packet delay bug. The progress of packet sometimes gets slowed going up or going down the stack. The easiest way to see this is to ping the stack and notice that every 4th or so ping is very slow. It's actually maddeningly deterministic. We've spent some time trying to track this down without much luck. If you're bored, feel free to try to track it down yourself. If you find this bug, we will crown you SuperDeveloper for the quarter and give you extra credit
- The socket module's support for minet_select() is quite buggy, so it is highly likely that select-based applications will not work.

We also appreciate constructive feedback and suggestions. We plan to use Minet extensively for teaching in the future. Your ideas, which we will credit accordingly, can therefore have a lot of impact.

Mechanics

- Your code must function as a tcp_module within the Minet TCP/IP Stack, as described in a separate, eponymous handout.
- Your code should be written in C or C++ and must compile and run under Red Hat Linux 6.2 on the machines in the TLAB. In particular, we will compile your code using GCC 2.95.2 and GNU Make 3.78.1, which are installed on the lab machines. You must provide a Makefile and a README. We will expect that running "make" will generate the executable `tcp_module` and that this module will meet the specification described in this document and in the "Minet TCP/IP Stack" handout. More details on handin will be available later.

Things That May Help You

- RFC 793 is essential.
- Chapter 3 of your book. Section 3.5 is a good introduction to TCP. Sections 3.6 and 3.7 are about congestion control. You should read them, but you do not have to implement congestion control.
- **Rick Stevens, "TCP/IP Illustrated, Volume1: The Protocols"**
- Doug Comer, "Internetworking With TCP/IP Volume I: Principles, Protocols, and Architecture"
- Rick Steven, "Advanced Programming in the Unix Environment"
- The handout "Unix Systems Programming in a Nutshell"
- The handout "Make in a Nutshell"
- The handout "The TLAB Cluster"
- The C++ Standard Template Library. Herb Schildt's "STL Programming From the Ground Up" appears to be a good introduction
- GDB, Xemacs, etc.
- CVS (http://www.loria.fr/~molli/cvs-index.html) is a powerful tool for managing versions of your code and helping you and your partner avoid stepping on each other's toes.