A6: Computer Networking (I)

Computer Systems 2018
Department of Computer Science
University of Copenhagen

Vivek Shah

Due: Sunday, 16th of December, 10:00
Version 1 (December 2, 2018)

This is the seventh assignment in the course on Computer Systems 2018 at DIKU and the first on the topic of Computer Networks. We encourage pair programming, so please form groups of 2 or 3 students. Groups cannot be larger than 3, and we strongly recommend that you do not work alone.

For this assignment you will receive a mark out of 4 points; You must attain at least half of the possible points to be admitted to the exam. For details, see the *Course description* in the course page. This assignment belongs to the CN category together with A7. Resubmission is not possible.

It is important to note that only solving the theoretical part will result in 0 points. Thus, the theoretical part can improve you score, but you *have* to do socket programming.

The web is more a social creation than a technical one. I designed it for a social effect — to help people work together — and not as a technical toy. The ultimate goal of the Web is to support and improve our weblike existence in the world. We clump into families, associations, and companies. We develop trust across the miles and distrust around the corner.

— Tim Berners-Lee, Weaving the Web (1999)

Overview

This assignment has two parts, namely a theoretical part (Section 1) and a programming part (Section 2). The theoretical part deals with questions that have been covered by the lectures. The programming part requires you to build a distributed chat service employing a combination of both client-server and peer-to-peer architectures using socket programming in C. The implementation task of the complete chat service would be spread over this assignment and the next one (A7). In this assignment, the programming effort relies solely on building the client-server portion of the architecture. More details would follow in the programming part (Section 2).

1 Theoretical Part (30%)

Each section contains a number of questions that should be answered **briefly** and **precisely**. Most of the questions can be answered within 2 sentences or less. Annotations have been added to questions that demand longer answers, or figures with a proposed answer format. Miscalculations are more likely to be accepted, if you account for your calculations in your answers.

1.1 Store and Forward

The answers to the questions in this section should not make any assumptions about specific protocols or details pertaining to the different layers. You can answer these questions after having read Chapter 1 of K&R book.

1.1.1 Processing and delay

Explain, within three or four sentences, the key reason for delay in typical packet switched networks, besides physical constraints such as the propagation speed of different transmission media.

1.1.2 Transmission speed

Consider the setup below in Figure 1. A DIKU student is using a laptop at home, browsing the `diku.dk` website. The upstream connection speed is 2 Mb/s from the DSL modem at home to the DSLAM[1].

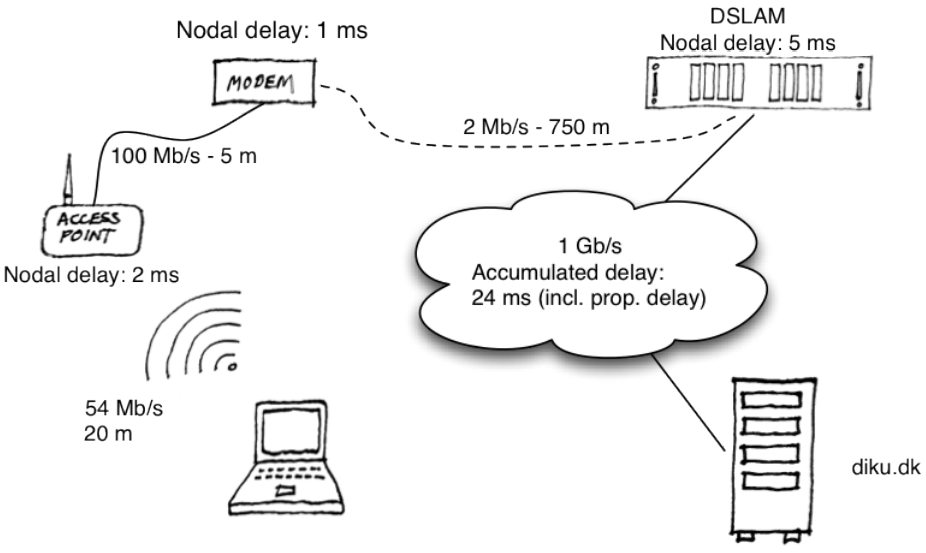

Figure 1: A typical DSL setup

[1] Digital Subscriber Line Access Multiplexer, used by Internet Service Providers to provide DSL connectivity over phone lines.

Part 1 Given the information in Figure 1, calculate the *round trip time* (RTT). For calculating propagation delay, assume that the propagation speed in all links visible is $2.4 * 10^8$ m/s and the queuing delay is contained in the noted node delays. You may leave out the propagation delay, but explain why, if you do.

Part 2 Assume 640 KB of data is sent to the `diku.dk` webserver, including any overhead. Assume that the server acknowledges the upload when all bytes have been transfered with a single packet. Calculate the total transmission time, given the RTT calculated above.

1.2 HTTP

1.2.1 HTTP semantics

HTTP employs a message format divided into header and body sections. The initial header of a request consists of a method field, a resource identifier field and a protocol version field. Likewise, the initial header of a response consists of a protocol version field, a status code and an optional message.

Part 1: What is the purpose of the method field in requests? How do `POST` and `GET` requests differ in practice?

Part 2: An additional (and mandatory) header field is the `Host` header. Why is this header necessary?

1.2.2 HTTP headers and fingerprinting

Part 1: One of the additional header fields is `Set-cookie` (for responses) and `cookie`. What is the reason for these header fields and to what degree may they be used as a unique identifier?

Part 2: The `ETag` response header works in conjunction with the `If-None-Match` and `If-Match` request headers to prevent unnecessary page fetches, if enabled. How can ETags work as cookies[2]?

1.3 Domain Name System

1.3.1 DNS provisions

Three of the most important goals of DNS[3] are to ensure fault tolerance, scalability and efficiency. Explain how these insurances can (and are) met in prac-

[2] You may want to consult section 13.3.2-3 and section 14.19 `http://www.ietf.org/rfc/rfc2616.txt`.

[3] As specified in RFC 1034 and RFC 1035, superseding RFC 882 and RFC 883.

tice. *Hint: Look at Figure 2.23 of the K&R book*[4] *and the resource record (RR) format — (Answer with 2-4 sentences.)*

1.3.2 DNS lookup and format

Part 1: Explain the advantages of the `CNAME` type records. Explain how DNS may provide simple load balancing among servers. *(Answer with 2-4 sentences.)*

Part 2: Many DNS servers, especially *root* and *top level domain* (TLD) servers, respond with 'iterative' replies to recursive requests. Explain the differences between iterative and recursive lookups and when and why recursive lookups are justified. *(Answer with 4-8 sentences)*

2 Programming Part (70%)

For the programming part of this assignment, you will build the client-server portion of the distributed chat service. The distributed chat service comprises of a peer client (`peer.c`) and a chat name server (`name_server.c`). The name server contains nicks and passwords of known users. The users must use the peer client to first sign in to the name server with their password and then register their addresses which users can lookup to engage in a peer to peer chat. This chat service would allow peers to directly chat with each other without going through the centralized name server.

In this assignment you do not need to implement the peer to peer chat mechanism, you just need to implement functionality in the peer client and the name server to allow log-in, check for other online users and log-out. For the purposes of this assignment, the peer client behaves like a regular client in a client-server architecture and does not engage with other peers. The code handout for this assignment consists of `peer.c` , `name_server.c` and their corresponding `.h` files. In addition, we have provided `csapp.h` and `csapp.c` files which contain the helper functions from the robust IO library and other helper functions contained in the B&OH[5] book. We have also provided a `Makefile` to make it easier for you to build the sources. *Use of the helper functions in csapp.h and csapp.c is optional. You can choose to ignore them if you wish. Ensure that your implementation can be compiled and run on the handed-out virtual machine.*

For the practical part of this assignment you are expected to hand in your C source code, test procedures showing validity (or invalidity) of your implementation and a document containing the answers to the questions as well as a short explanation of how you implemented your code (if your code has any shortcomings you should document them here). *The weights for the programming task and the report detailing your implementation are 45% and 25% respectively. You are* ***not*** *allowed to use any external libraries other than the standard C library for*

[4] Computer Networking: A Top-Down Approach, James F. Kurose and Keith W. Ross, Pearson, 7th and International Edition

[5] Computer Systems: A programmer's perspective, Randal E. Bryant and David R. O'Hallaron, Pearson, 3rd and Global Edition

this assignment. You can use the helper functions that have been handed out with the assignment sources.

2.1 The Peer Client

The peer client uses a simple IRC[6] like command functionality. The client should allow the user to perform the following interactions:

2.1. Login - The user should be able to log-in to the chat name server specifying a nick, password[7] and her address information i.e., the IP address and port on which the peer client would be listening for peer chat messages. For this assignment, you do not need to implement the code for the peer client to listen for messages from other peers (that will be handled in the next assignment). Incorrect nicks and passwords should be rejected. The syntax for the command is :

```
/login <nick> <password> <IP> <Port>
```

2.2. Lookup - The user should be able to query the chat name server for the address information of another nick or of all nicks currently signed in. If no argument is provided for lookup, then the information of all currently signed in nicks should be provided. If lookup is invoked with an incorrect nick i.e., a nick unknown to the name server then an error should be flagged. Only logged in users should be allowed to perform a lookup. The syntax for the command is either :

```
/lookup <nick>
```

or

```
/lookup
```

2.3. Logout - The user should be able to log out of the chat name server and de-register her address. Only logged in users should be able to log out. The syntax for the command is :

```
/logout
```

2.4. Exit - Terminate the client program.

```
/exit
```

You should implement the above mentioned functionality in the file `peer.c` and `peer.h`. You are also allowed to create additional files if it helps to make your code modular. A sample interaction with the peer client program is outlined in Figure 2.

The peer client should not try to connect to the name server before the user gives it the appropriate input. Command line arguments are used to provide the peer client with the IP address and port of the name server.

[6] https://en.wikipedia.org/wiki/Internet_Relay_Chat
[7] You do not have to consider security in your login procedure.

```
$ ./peer 192.168.1.42 3456
/login bonii secret 192.168.1.41 8181
You are now logged in.
/lookup bonii
bonii is online.
IP: 192.168.1.41
Port: 8181
/lookup xyz
xyz is not a valid user.
/lookup
1 user online. The list follows
Nick: bonii
IP: 192.168.1.41
Port: 8181
/logout
You are now logged out.
/exit
$
```

Figure 2: Sample client interaction

For this assignment, you need to devise and implement the protocol (messages and their handling) that should be performed by both the peer client and the name server.

2.2 The Name Server

The name server should implement the server-side of the protocol you devised to support the client interactions outlined previously. When the server starts, it should be able to infer the nicks and their corresponding passwords of users who are allowed to log in to the chat name server. For simplicity, you can define them in some struct in your `name_server.h` or `name_server.c` file or you can read it from a file if you wish. Be sure to mention the mechanism you chose for this in your report. *You should ensure that the server can handle and interact with multiple clients and does not have to wait for a client to log out to respond to another client.*

You should make sure the chat name server sets up the needed sockets at initialization (hint: make the address of the listening socket reusable. That way you will not have to wait for the socket to time out if the server crashes). When the chat name server is running it should listen for new peers trying to connect to the chat service and existing peers sending queries to the name server. Furthermore, it will need to keep an eye out for dead sockets. If a client program is killed or a socket dies, the chat name server should detect this and remove that socket from memory and close it. You should implement the chat name server functionality in the file `name_server.c` and `name_server.h`. You are also allowed to create additional files if it helps to make your code modular.

2.3 Report

In addition to implementing the programming part, you should also answer the following questions in your report.

2.1. Describe how to compile and run your code. How did you test your implementation? (Note: Automated test cases are not a strict requirement for this assignment; detailing the testing procedure or methodology in the report is sufficient. Test your implementation by running the name server and peer client on the same machine before different machines since some networks disable certain forms of traffic that can affect your testing procedure.)

2.2. Explain the protocol that you used to implement the functionality. Here you should document the type and formats of messages exchanged between the peer client and the chat name server and the events that followed on sending and receiving the messages.

2.3. Discuss the non-trivial parts of your implementation and your design decisions (if any). Explain *why* you wrote your code the way you did, not merely *what* it does. *Purpose is more important than mechanism.* Remember to reflect on the shortcomings (if any) of your implementation and how they can be fixed.

2.4. The current design of the distributed chat service uses a centralized name server. Is this a good idea? What advantages and disadvantages are there of running a centralized name server? How could we change the service so that the name server would be distributed?

Note: Remember to provide your solutions to the theory questions in the report.

Submission

The submission should contain a file `src.zip` that contains the `src` directory of the handout, with a modified `peer.c`, `peer.h`, `name_server.h`, `name_server.c` and `Makefile`. Furthermore, it should include any new files or test programs that you may have written.

Alongside the `src.zip` containing your code, submit a `report.pdf`, and a `group.txt`. `group.txt` must list the KU ids of your group members, one per line, and do so using *only* characters from the following set:

$$\{\texttt{0x0A}\} \cup \{\texttt{0x30}, \texttt{0x31}, \ldots, \texttt{0x39}\} \cup \{\texttt{0x61}, \texttt{0x62}, \ldots, \texttt{0x7A}\}$$

Please make sure your submission does not contain unnecessary files, including (but not limited to) compiled object files, binaries, or auxiliary files produced by editors or operating systems.