Assignment A6
CompSys

B. Alix, K. A. Madsen, C. S. Hansen

20. marts 2019

1 OBS: Resubmission

Genafleverings delen bliver skrevet på dansk. Den starter på side 6.

2 Introduction

This assignment is the first topic about Computer Networks. The assignment is divided up in two parts, namely a theoretical part and a programming part. Where the theoretical part deals with questions that have been covered by lectures. While the programming part will be about building a distributed chat service employing a combination of both client-server and peer-to-peer architectures using socket programming in C.

3 Theoretical part

3.1 Store and Forward

3.1.1 Procession and delay

There are three reasons to why a delay might occur in a typical package switching: the router must first buffer the all the received packets bits and when all bits are received, then the packet can be transmitted. The second reason is a queuing delay, if many packets are waiting to be processed at the router, then the queue can be long. Lastly the third reason is a transmission delay. This is the amount of time the router needs to push all of the packets bits into the link. [1]

3.1.2 Transmission speed

Part1

Using Figure 1 in the assignment, we are asked to calculate the round trip time (RTT). We are given in the assignment that we should assume that the propagation speed in all links visible is 2.4 x 108 m/s and the queuing delay is contained in the noted node delays. Therefore in our equation we do not have to add d_{queue} and can ignore the propagation delay as it's so small.

[1] K&R pg. 51, 64

- d_{proc} = process and queuing delay
- d_{trans} = transmission delay
- d_{prop} = propagation delay

$$d_{endtoend} = (d_{proc} + d_{trans} + d_{prop})s$$

[2] We need to transport L bits on the DIKU server.

Here we use only the nodal delays.

$$\begin{aligned} RTT &= 2(24ms + 5ms + 1ms + 2ms) \\ &= 64ms \end{aligned} \tag{1}$$

Part2

Here we are asked to compute the total transmission time. Here we multiply the package size of 640KB and then sum up the transmission delays for each node and then add the RTT we calculated in the previous problem.

$$640KB = 640kb \cdot \left(\frac{1}{54000kb/s} + \frac{1}{100000kb/s} + \frac{1}{2000kb/s} + \frac{1}{125000kb/s} + 0.064s = 0.407s \right)$$

Converted gb/s to kb/s, mb/s to kb/s, and ms to s

and back to ms $= 0.407s = 407ms$

$TTT_{640KB} = 407ms$

(2)

3.2 HTTP

3.2.1 HTTP semantics

Part 1

The method field is responsible for specifying actions and holds different methods depending on what action should be taken. Among others, two of these methods are POST and GET. GET is used when the user wants to access an object in the browser. The POST method accepts user input and sends it onward to the server. An example could be a when a user searches for something with a search engine.

[2] K&R pg. 69

Part 2

The HOST header is necessary to handle the different objects or services hosted on one server.

3.3 HTTP headers and fingerprinting

Part 1

The field Set-cookie generates an identity number for the user who is visiting that particular web-page. Cookies are, according to the book, "allow sites to keep track of users"(Computer Networking pg. 136). As each user who visits "www.example-random.random"it could be surmised that cookies are a great unique identifier, but as the cookies are only useful up until the user deletes them, they are actually unreliable.

Part 2

The ETags are sent by the browser to the server and as the ETags are stored on the server and not locally on the users end, these could be used as an actual reliable way to track users uniquely.

3.4 Domain Name System

3.4.1 DNS provisions

Probably the most important principle is the hierarchy, which is responsible for facilitating scalability and efficiency. By implementing a hierarchical system that maintains a central root, the DNS can scale productively. In conjunction with caching the hierarchy system also has a certain level of fault tolerance.

3.4.2 DNS lookup and format

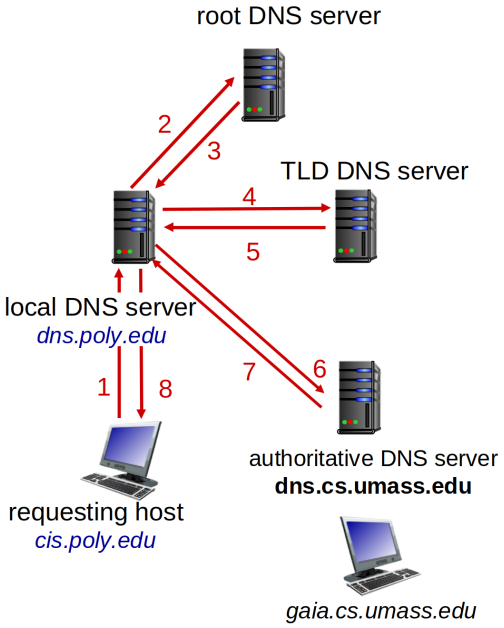
Part 1

The usage of CNAME records enables multiple services to run from the same shared IP address. A way that DNS can provide simple load balancing among servers is that when multiple servers share a common name then the incoming requests, i.e. workload, can be shared across said servers.

Part 2

What is the difference between iterative and recursive lookups?

To explain the difference between iterative and recursive lookups, we will use the following example (see figure 1 from the book[3]) to explain it. In this example the DNS servers makes use of both **recursive queries** and **iterative queries**. We see that the query sent from `cse.nyu.edu` to `dns.nyu.edu` is a recursive query, since the query asks `dns.nyu.edu` to obtain the mapping on its behalf. Recursive DNS queries occur when a DNS client requests information from a DNS server that is not able to resolve the requested query itself, then it forwards the query to subsequent DNS servers until a definitive answer is returned to the client or until the query fails.



Figur 1: DNS Server: Recursive and iterative queries

While the three subsequent DNS servers are iterative, as all of the replies are directly returned to `dns.nyu.edu`[4]. Iterative DNS queries in a DNS serer is queried and returns an answer without quering other DNS servers, even if it cannot provide a definitive answer.

When and why are recursive lookups justified?
As explained earlier, recursive DNS lookups occur if a DNS client requests information from a DNS server that it is not able to resolve the requested query itself, then it forwards the query to subsequent DNS servers until it gets an answer, or until the query fails.

[3]Kurose & Ross, Computer Networking: A Top-Down Approach, Global Edition, 2017, page 161
[4]Kurose & Ross, Computer Networking: A Top-Down Approach, Global Edition, 2017, page 162

4 Programming part

For peer.c we have made us of the helper functions in the c libraries, especially the rio library. First we handle the initial buffering i.e opening a connection to the server when a login request comes in. How the server is to handle the login, logout and lookup; these are contained within the Main part of peer.c. By doing such our goal was to keep the instruction flow as smooth as possible.

In relation to server

5 Testing

6 Discussion

6.1 Non-trivial parts of the implementation

6.2 Ambiguous formulations

In the peer client it is not really clear that exit and logout should be two different commandoes, as they partly do the same.

We think that the whole description of how to make the name_server was very confusing

7 Conclusion

We have done our best to answer the theoretical part, and doing this process we learned alot about the application layer using chapter 2 from the book K&R. We have also done our best to finish the programming part, but we did not quite finish it, unfortunately. This happened because, we started too late this time, had a partly bad schedule and we ran out of time. However, from what of it we have made, we have gained a greater understanding of how to use socket programming in C.

8 ny Programming part

8.1 Implementation

For at forenkle koden og gøre den mere læsbar, og modulær, har vi valgt at hver interaktion mellem name_server og peer programmerne som et par af funktioner som har samme navn med adskildelse mellem server og peer med hhv. stort og lille start bogstav. Det er alle at finde i help.c og help.h. Dette gør det muligt at skifte enkelte dele af begge programmer ud uden at det giver fejl, så længe begge programmers funktions par overholder samme besked format. Vi kunne have valgt at hver funktion skulle tage et format som argument, og herved gøre koden endnu mere modulær, men valgte dog ikke at give denne mulighed. Vi har desuden lavet tre hjælpe funktioner Send, Readline og Source_dump.

Hver funktion tager en file discriptor og et char array (string) som input,[5] i hvert tilfælde er file descriptoren output kanal, som med sit generiske interface gør det muligt både at skrive til filer, sockets og pipes, hvilket gør det nemt at test funktionernes adfærd. Char arrayet er argument kontaineren. lookup tager en ekstra file descriptor som argument, denne er både som hjælp til at teste, men også brugbar i A7, da det gør det muligt for peer, at bruge */lookup <nick>* til at finde IP og port ved at kalde lookup internt i forbindelse med at sende en besked til en anden peer. Da output så nemt kan blive omdiregeret.

name_server bruger filer som lager til peer info, så som *bonii.off* og *bonii.on*. Hvor file ending er indikator for h.h.v. om en peer er online og om hvis ikke om denne bruger eksistere. Formattet for de to filer er

For <nick>.off

NICK: <nick> PASS: <password>

for <nick>.on

IP: <host> Port: <port>

Send and Readline funktionerne er wrappers af csapp.c funktioner, hvor Send selv finder længden af de der skal sendes og sætter linjeskift. Readline fjerner buffering fra brugerens fokus, men er ellers bare en wrapper for Rio_readlineb. Source$_dumpharsomnavnetantydertilforml, attagerindholdfraenfiledescriptoro$

[5] på nær Source$_dumpoglookupsom$

8.2 Testing

For at køre test, skrive *./test.sh* i terminalen sat i dic *src*.

Der er inkluderet test for Send, Readline, Source_dump, (login, Login), (logout, Logout) og Lookup. Vi har ikke nået at teste lookup samt name_server og peer.

Alle test er lavet udfra den konkrete implementation af nameserver og peer. Det betyder, at flere grænsetilfælde ikke bliver testet, men antaget at de bliver filtreret fra i name_server og peer før kald af diverse funktioner. Alle test burde køre fejl frit. Exit funktionen er ikke blevet testet da den kun kalder logout og Send som i forvejen er testet, så den virker under antagelse af at Send og logout virker.

9 Appendix