Compsys - A3

Christian R (srw812), Nicolai N (dwx635) og Andreas J (cmn945)

November 21, 2021

Theoretical

Store and Forward

Processing and delay

There are nodal processing delays and queuing delays. The reason for processing delays are amount of cycles in the CPU needed to look at the packets header and decide what to do with the packet and then send it to queue. In queue we are delayed with 0 if there are no earlier arriving packets in front of us. The more traffic before us means longer queue times.

Transmission speed

Part 1
To find the RTT, we follow the formula: RTT = 2· propagation delay.
We accumulate all the distances: $750 + 5 + 20 = 775$ m. We divide this with the propagation speed that is $10^8 \cdot 2.4$ and add the nodal delay $0.001 + 0.024 + 0.005 + 0.002 = 0.032$.

$$2 \cdot (\frac{775}{10^8 \cdot 2.4} + 0.032) = 0.0640s$$

Part 2
We accumulate every transmission cost to find the total transmission time.

$$\frac{0.64 \cdot 8}{54} + \frac{20}{10^8 \cdot 2.4} + 0.002 + \frac{0.64 \cdot 8}{100} + \frac{5}{10^8 \cdot 2.4} + 0.001 + \frac{0.64 \cdot 8}{2} + 0.005 + \frac{0.64 \cdot 8}{1000} + 0.024 = 2.7431s$$

HTTP

HTTP semantics

Part 1 The purpose of the method field in HTTP is identify the object by the URL. The GET method is used by the client to retrieve data that is identified by the URL, but the entity body stays empty here, while the POST method requested web page depends on the entity body, this entity is defined by the user input on the web page.
Part 2
The Host header method requests where the data is being sent to. This consists of an IP address and port.

HTTP Header and fingerprinting

Part 1
Set-cookie header the server responding to clients request, while a cookie is used to create a unique id for the client. This allows the server to store clients data and using set-cookie header allows the client to send cookies back to the server.

Part 2
ETags are response headers that cache data from users on servers and otherwise work similarly to how cookies work.

DNS

DNS provisions

DNS fault tolerance is met in practice by attempting to contact a DNS server, if there is no response you move on to the next on the list. Scalability is ensured through the hierarchical nature. Since data can flow through many different intermediary devices, there will be no single point of failure. Adding new servers to the system to help is also easy. DNS also uses caching. Local servers then cache the data and store it for an amount of time, ensuring efficiency.

DNS lookup and provisions

Part 1
Sometimes domain names are connected to the same IP address. CNAME allows a device to have unlimited number of CNAME aliases. This allows servers to have several sub domains pointing to your main domain.
Part 2
Iterative lookups work by searching local DNS servers directly, while recursive DNS lookup communicates with several DNS servers to track down an IP address and return it to the client.

Report

Introduction

In this assignment we were to learn about network programming. We have been given template code and we were required to fill out missing parts of a file transfer service using socket programming. As the first of a two part assignment, we were to implement the client part of a P2P network.

Protocol

The cascade protocols ensure that the requests are written the same. This prevents data loss, delay and traffic. If cascade didn't have a header, then the tracker would have no way of returning the IP address, the size of the information or the hash numbers. Dividing the data into bite sizes pieces and using hash checks to ensure the order of the data.

Implementation

first we implemented the parsing of the .cascade file. From the header we could gather most of the information based on the cascade file format provided in the assignment, only modifying the information there as we wanted to convert the provided file hash from network byte order into host byte order. From this we then read through the rest of the file, in order for us to collect all the the individual blocks so they are more workable
next we checked for any existing blocks already present on our system, so we don't have to download them twice, this also provides the functionality that you can stop the download process and can pick up later where you were, rather than start all over when you already have some of the files on your system.
We then implemented finding the SHA256 hash of the cascade file so we could use it to send to the tracker, allowing it to find us the peers which have the file we want to download, which is what we implemented next to get back a list of peers which can serve us the file
Lastly we implemented so the user can download from the peers which have their file from a queue of the files which were found earlier to not yet be present on the system

Design testing

Unfortunately our design is by no means complete. We have to simply admit that we started much too late with the assignment and therefore have hardly had the time to properly implement the functionality, let alone test it in any real way, and therefore we cannot provide any real examples of

testing since our implementation barely even compiles, providing a multitude of error messages that have yet to be fixed.

Conclusion

If we had time to go into detail about every shortcoming of our project then we'd use it to fix our program rather than listing them all here. I'll leave it at saying that, while everything has, in general terms, been implemented, all of the details have yet to be ironed out, both in terms of fixing major program-stopping bugs, but also after that is fixed we'd need to fix the smaller scale bugs, which are no doubt present, that don't cause the program to crash, but messes with results.