Communication networks
Project

Submitters: Barak Finkel 206491714
Daniel Schneiderman 314790197

Date: 13.03.23

OVERVIEW

In our project, we have simulated various types of network communications a client makes throughout their existence.
The communications that we have simulated vary from IP assignment requests via DHCP protocol, DNS requests in order to retrieve an IP of a distanced website, and a communication between the client and an application which sends a file to our client.

We chose to build an application that receives HTTP request from our client (more details later on), and redirects the clients to a location that contains a file that the user requested via the HTTP request above.

The file downloaded is being sent in a custom protocol based on UDP, but has an added header layer that provides credibility to the data sent.

The assignment consists of the following files:

- dhpc_server.py – The DHCP server that serves our client by assigning him to an IP.
- dns_server.py – The DNS server which listens to query requests and sends back responses accordingly
- mystery_song_server.py – The proxy server which receives HTTP requests for another IP and redirects the client there.
- RUDP_Server.py – The server from which the file is actually being sent.
- client.py – The client which makes requests towards all of the above.
- RUDP – A custom layer added to UDP packets with data purposed to ensure their credibility.

A few more details about our programming environment, tools and other files:

- Operating System – Ubuntu 22.04
- Programming Language – Python
- Wireshark Recordings – .pcapng files (with names specified to imply the delay times of packets)
- Running Example – A video showcasing how to execute the files via the terminal, named RunningExample.mp4

The next pages will cover details over the communication between the client and each of the servers.

NOTE – Specific details over the actual implementation are given in the form of quotas in the code. Please examine them for further explanation over the implementations.

Client vs DHCP Server

In our first part of the whole program, we simulate a communication between a DHCP Server and our client.

The DHCP Server in concept is a router, responsible for listening to broadcasts from users in it's LAN that are missing an IP address and assigning them with one. Such address is needed in order for them to be able to communicate with other devices also from outside of the LAN.

The Communication:

As implied above, the client first broadcasts a message through port 68 which is specifically used for clients in DHCP communications.

The server listens to broadcasts on port 67, which is parallelly used for DHCP servers.
When he receives our client's broadcast, he searches through his array of IP Addresses and checks if any are available. If so, he sends an offer message back to the client.
Else, he simply ignores the broadcasts and continue listening to future clients.

In case the server sent an offer back to the client, the client reads through the message, and sends a request message containing the IP it was just offered.

The server then checks again if that IP is in it's array of IPs and that it's available.
If so, he assigns the client with it and sends back an ACK packet to the client containing the DNS server's IP to be able to communicate with other devices throughout the internet.
Else, he sends a NAK packet informing the client that his request is rejected.

!!! It's important to note that the server checks the availability of the IPs with a timestamp attached to each IP at the moment it's assigned to a client, representing the expiration time of that IP assignment.

Returning to our communication, if the client gets an ACK back from the DHCP server, he saves the IP he's given and the DNS' IP. Then, he will operate with his new personal IP up until the expiration date. When that expiration date arrives, he will be forced to shut down.

Client vs DNS Server

Next, we simulate a communication between a DNS Server and our client.

The DNS Server is addressed to by our client in order to receive IP addresses of other devices throughout the web. The DNS server is crucial middle-point for our client, that would otherwise would not be able to communicate with anyone aside from the DHCP Server.
Simply put, the DNS server listens to client requests asking for a URL address, and sending them back a domain IP that is responsible for that URL.

Note – The DNS server listens to DNS queries on a dedicated port, which is 53.

The Communication:

In comparison to the DHCP server, the communication between the DNS Server and our client is a bit more simple.

The client requests a URL address from the DNS server via a query packet send through the DNS protocol.

The DNS server, that listens to query packets, first searches through a cache of previously saved addresses. This cache is maintained in a way that every address is saved for an hour and then erased from the cache so that it will only contain relevant information.

If the URL was found in the cache, the server sends a response to the client containing the domain IP of the matching URL.

In concept, if the DNS server doesn't find the relevant IP in it's cache, he will send multiple DNS queries gradually to more specific domains, but we have not implemented that in our project. Therefore, we assumed that the wanted IP was already existing in the cache.

Then, the client receives the domain IP, and addresses it.

Client vs Mystery Song Server

In this part we will go over the concept of our application server.

The Mystery Song Server is an app in which a client makes a general request of a song, but not with a song name. Instead, the client is only providing the server with a song genre and the language the song is sang in, and in return he receives a URL from which he downloads the "mystery song".

Basically, the server owns a list of songs, their genre and their language and their unique URL address from which they could be downloaded from.

<u>The Communication:</u>

The communication is made over the HTTP protocol.

The client sends the server an HTTP GET packet, containing his desired genre and language of the "mystery song".

The server listens to requests, and as soon as he gets one, he analyzes the list of the songs he owns, and filters matching songs matching the request.
Afterwards, he randomly chooses a song from the filtered list, and sends it's URL back to the client with a 301 packet, containing the URL.

If the server did not find a song matching the request, he sends a 404 packet back to the client, implying the request could not be answered.

The client then addresses the URL given to him.

Client vs RUDP Server

In this last part of the program, our client addresses the server that holds the song requested from the Mystery Song Server.

RUDP (aka, Reliable UDP):

The sending and receiving of the file is done over an RUDP protocol.
More specifically, we use the base UDP protocol to transfer chunks of the song's file to the client, but each packet is sent with an additional custom layer made by us containing the details below:

- Sequence Number – The serial number of the packet sent.
- Window Size – The number of packets sent before receiving an ACK back.
- Start Num – The first packet sent within a window of packets.
- End Num – The first packet sent within a window of packets.
- Flags – a Byte indicating if a packet sent is an ACK / SYN / FIN packet.
- Payload Size – The size of the file's chunk sent within the packet (in bytes).

All of this information sent within a packet helps the client and the RUDP server to determine whether the data was sent properly or not – giving us the reliability we need that are given from other transportation protocols such as TCP.

The Communication:

1) First is a 3-way handshake between the client and the server.
   - The client requests to connect with the server (With a SYN packet)
   - Then, the server acknowledges the request to connect, and also requests to connect with the client. (With a SYN-ACK packet)
   - Lastly, the client sends an ACK packet to the server, also containing the URL of the song he requests, and also the initial size of the window of packets it's ready to receive.

2) Afterwards, The server starts sending windows of packets to the client.
   The windows have their size adjusted throughout the communication to match the capabilities of both the server to send that packets, and the client to receive them.

   Whenever a window of packets is received successfully by the client, he sends an ACK packet back to the server. Before sending another window of packets, the server either multiplies the window size by 2 or increments it, depending of whether the size of a window is above or below a threshold (initially set to be 16 packets).

When the window is not received successfully, the client sends a NAK packet back to the server. Before sending another window of packets, the server divides both the window and the threshold by 2, and trying to send a window of packets starting from the same sequence number as before.

3) When the server finishes sending the file, a 3-way handshake starts again, this time for finishing the communication.

   - The server sends a FIN packet to the client, informing him that he's done sending the file.
   - The client then sends him a FIN-ACK packet, informing the server that he acknowledges his request to end the communication, and requesting to do so as well.
   - Lastly, the server sends an ACK packet to the client, ending the communication.

Note – All of the above demonstrates a properly made communication between the 2 parties. But, if at any point, after 3 attempts of either waiting for a packet from the other party, or 3 failed attempts of sending a window of packets, the communication is ended.

If less than 3 attempts failed, the written above in step 2 is executed.

Packet Loss Handling:

Our RUDP server supports packet loss by having mechanisms that recognize it, and providing a solution within run-time.

Flow Control is achieved by 2 main factors:

1) The client sends the initial window size he requests at the initial connection to the RUDP Server. This lets the server know how many packets the client can receive at first.
2) If the client failed to receive all packets in a window (from the starting packet and up until the last), it sends the server a NAK packet, requesting him to re-send a window of packets again.

Congestion Control is achieved by 2 main factors:

1) The server sets a threshold, that stops the acceleration of the inflation of the window size. Simply put, after multiplying the window several times, the server instead only increments the window size by 1 for each window so that they would not overwhelm the client.
2) Whenever the client failed to receive all the packets in a given window, the server divides the threshold and window size by 2, and tries sending a smaller window next to assure that the next few or more windows will be sent and received properly.

Latency Issues handling:

Our RUDP Server and client supports latency issues by having small sleep() intervals between sniffing and sending packets on each side.

Moreover, while one party expects the other to send a packet, we have set a timeout of 3 seconds on each sniff method in order to give a solid opportunity for a packet to arrive before either trying to send packets again or terminating the connection.

This helps both the client and the server process the packets and send responses at reasonable rates while also supporting latency issues that might occur.

Simulation of Packet Loss:

In our code, we have simulated packet loss with the reduction of sleep intervals between after each packet send. The relevant sleep() methods are marked with '#########' quotas in the codes of the Client and RUDP server.

The results of each reduction are shown within .pacp files added in the directory 'ProjectRecordings'.

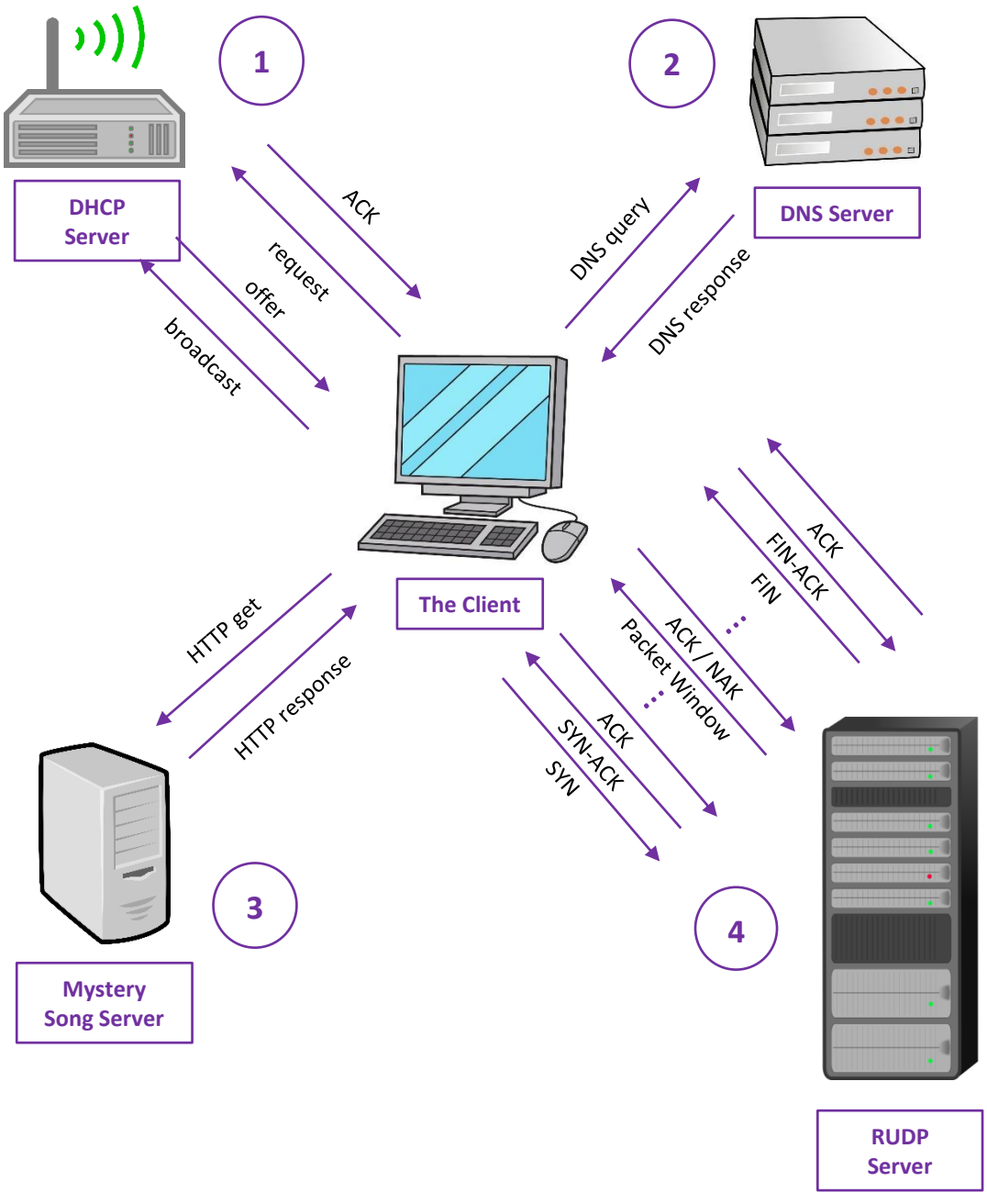
Here are the results for each different sleep time interval:

- 0.04 seconds – The windows were sent perfectly, without any packet loss.
- 0.03 seconds – Seldom, the packets were not sent properly. Only when a window reached a size of about ~15 packets, there were 1 or more packets lost.
- 0.02 seconds – Periodically, the packets were not sent properly. Only when a window reached a size of about ~8 packets, packets were lost.
- 0.015 seconds – At this point, it was near impossible to send the packets. Very often packets were lost and at a non-measurable rate.

From this we concluded that our CC and FC algorithms could support up to periodical packet loss, but when there were major losses, it took a very long time to send the file.

Note – When trying to reduce the sleep interval to 0.01 seconds, the file always failed to be sent completely. Therefore, we haven't added a sample of that experiment.

The Program's Communication Diagram



Theoretical Questions:

1) List 4 differences between QUIC and TCP:

   Answer:
   1) While TCP's reliability is promised from data in the transportation layer, QUIC's reliability is promised from data in the app layer.
   2) While TCP is making the initial connection via a 3-way handshake, QUIC doesn't require it.
   3) While the encryption of data in a TCP based communication is made only after the initial connection is made, in QUIC it happens at the moment of the connection – directly because QUIC starts sending data while creating the connection.
   4) While TCP handles sending files within a singular stream, QUIC can use multiplexing in order to handle multiple streams to send files. This could be done in order to split a large file into chunks and send each in a different stream, or send a different file within each stream.

2) List 2 differences between Vegas and Cubic CC protocols:

   Answer:
   1) While Cubic accelerates the inflation of the window size of packets up until a set threshold and only then reduces it, Vegas periodically sets a threshold based on measuring response time intervals of the packets sent throughout the communication.
   2) While Cubic detects packet loss based on information gathered for it by the routers in the way of the packets transportation, Vegas measures congestion loss by itself using his own timers to determine packet loss.

3) Explain what is the BGP Protocol, what's the difference between it and OSPF, and if it uses short paths.

   BGP is a routing protocol responsible for finding the best route to transfer data from 1 point to the other. The difference between BGP and OSPF is that BGP is mainly used in order to find a path between servers across different domains, while OSPF is responsible for finding the shortest paths between 2 end-points in a domain.

   BGP doesn't always prioritize the shortest path, because it also measures other variables such as the congestion of a route – and will calculate a priority for a route accordingly.

4) Provide data over your project in the following chart, and explain:
   1) How would the messages change if there was a NAT between the client and the servers?
   2) How would the messages change if you used QUIC instead.

   Answer:

Application	Port Src	Port Des	IP Src	IP Des	Mac Src	Mac Des
DHCP	68	67	0.0.0.0	255.255.255.255	found in run-time	found in run-time
DNS	20714	53	10.0.0.13 10.0.0.14 10.0.0.15	10.0.0.11	found in run-time	found in run-time
MysterySong	20714	30197	10.0.0.13 10.0.0.14 10.0.0.15	127.0.0.1	found in run-time	found in run-time
Rudp	5000	5001	10.0.0.13 10.0.0.14 10.0.0.15	10.0.0.100	found in run-time	found in run-time

   1) Assuming the client is in the NAT's space, the servers would need to address the client's public address, which is necessarily different that the real address that the NAT holds.
   2) If we used QUIC, we wouldn't need a 3-way handshake. The reason is that QUIC handles the initial connection immediately without needing to send SYN and ACK packets.

5) Explain the differences between DNS and ARP

   Answer:
   There are 2 main differences:
   1) While DNS returns IP addresses, ARP returns MAC addresses.
   2) While DNS return addresses to locations outside of a sub-net, ARP returns addresses from within it.