Iker Garcia

2315487

11/22/2024

Video Streaming Research Project

Introduction

In the Computer Network field, Video Streaming is one the most complex tasks to do. Aside from the difficult task of processing video for both clients and servers, finding the correct package size, knowing when to lower video quality depending on the state of the network and ensuring that errors in the delivery of packages affect the user's experience the least possible are all laborious tasks that require a lot of attention and knowledge in the field. For this project, we will create a basic video streaming service that runs on a local network, use this service as a tool for network calculations (such as latency) and test how the service is affected by other regular tasks a network has to do aside from video streaming (i.e. web browsing, uploading files, etc.).

Hypothesis

Due to the video streaming service being run locally, the networks latency will be extremely low. However, running other tasks in parallel (like web browsing) will have a significant effect on the latency of the videos streaming service. In addition, dividing the video into smaller packages will have a significant effect on the latency and the client's experience when watching the video.

Data Source and Analytical Approach

To create the Streaming service, we used Abdisalan's Mohamud code as a basis. This code consists of HTML/JavaScript project which has a frontend and a backend. The front end is a basic HTML page which outputs a video, with the basic pause, mute, forward, etc. buttons. The frontend is a JavaScript file which receives a JSON header with the last piece of video the client has received. It then calculates the size of the package based on a Min operation which compares the addition of the position of the client plus the chunk size (in this case, 1MB) or the total size of the video (in case the video has less than 1MB left to finish). It then parses the video with the calculated length and sends it back to the server with HTTP Status 206, which is the code used for partial content being sent. We modified this code to calculate the server processing time, saving the time when the request got to the server and subtracting it to the time when the 206 HTTP response was sent back.

Additionally, we used Google Chrome's Inspect menu, which permits users analyze all the requests sent and packages received while running a web application. It lets users see the amount of time a request took to be answered by the server, and the amount of time it took to download it to the client.

To get the network's status, we used several metrics from both the server and the client (all these metrics where captured in milliseconds):

- Client waiting time: the amount of time elapsed between the client sending a request and receiving an answer (without accounting for the time the client took to download the answer)
- Server Processing time: Time elapsed between the server receiving the request form the client, parsing the video file and sending the video package to the client
- RTT: Round Trip Time, or the amount of time the network took to send the request to the server and send the response back to the client (without the server's processing time)
- Latency: Time elapsed on a single trip. Half of RTT.

Procedure

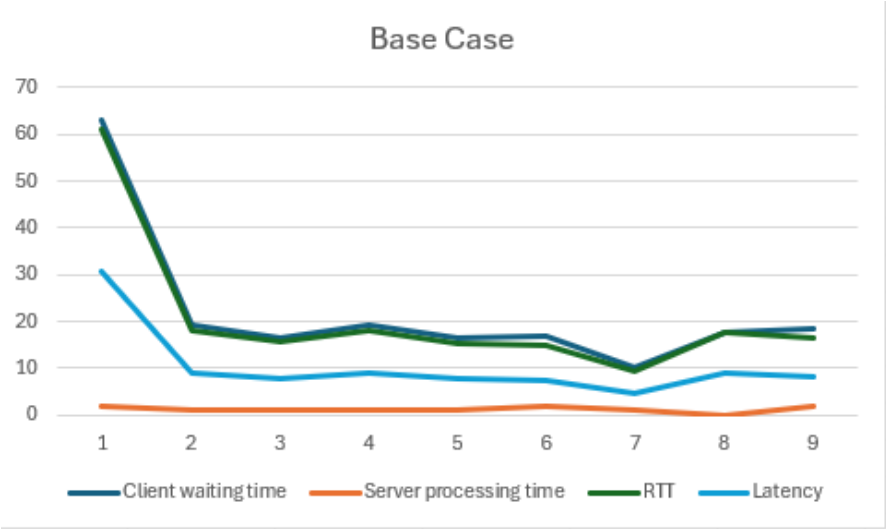
We began by running the JavaScript code on port 8000. On a different laptop (connected to the same network), we opened the Chrome Web browser and inserted the Server's ipv4 address and the port number (192.168.1.209:8000). After the Web browser successfully found the server in the local network, we opened the "Inspect" menu and began capturing all the request and packages being received. We then played the video and made sure both the server and client where capturing data correctly. After the video ended, we finalized the server's code and stopped recording packages on the client's side. We saved the results from both parts and deleted the cash memory on the client to evaluate the program again.

Results

We evaluated the program 5 separate times, while trying to saturate the network in diverse ways:

Base case

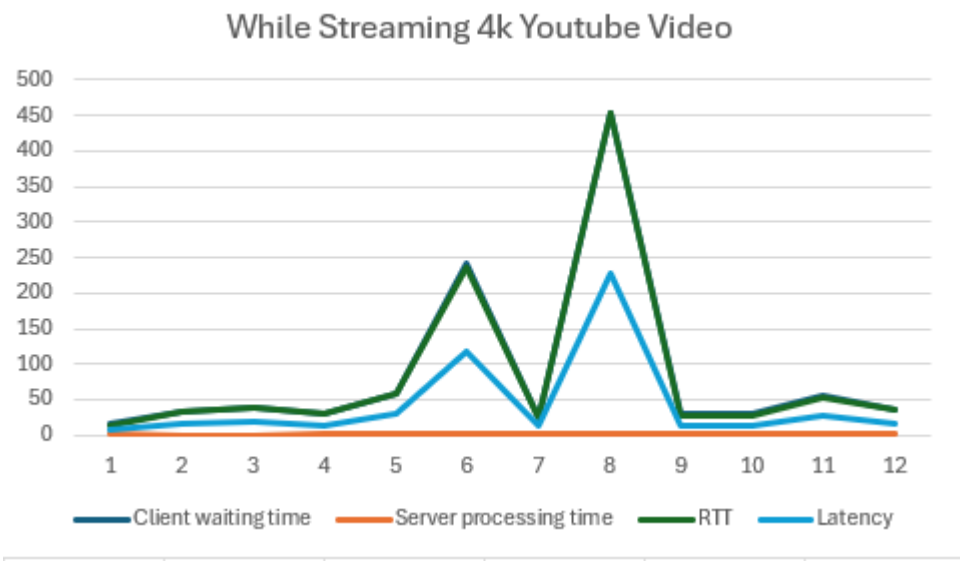
	Client waiting time	Server processing time	RTT	Latency
	62.98	2	60.98	30.49
	19.23	1	18.23	9.115
	16.57	1	15.57	7.785
	19.18	1	18.18	9.09
	16.38	1	15.38	7.69
	16.94	2	14.94	7.47
	10.28	1	9.28	4.64
	17.54	0	17.54	8.77
	18.28	2	16.28	8.14
Average	**21.93111111**	**1.222222222**	**20.7088889**	**10.3544444**



We got on average a latency of 10 milliseconds (ms), which is particularly good number considering some sources consider having a latency of less than 50 ms a good number for WAN networks. Client waiting times, RTT and Latency decreased in value as the video continued playing, this could be since, during the first request, the network had to find the server the client was requesting, whereas as the video continues, the network already had the route saved in cache. Server processing time remains the same regardless of the state of the network.

While streaming 4k YouTube video on the server

	Client waiting time	Server processing time	RTT	Latency
	16.85	3	13.85	6.925
	32.65	0	32.65	16.325
	39.74	0	39.74	19.87
	29.82	1	28.82	14.41
	59.26	1	58.26	29.13
	240.36	3	237.36	118.68
	24.91	1	23.91	11.955
	454.02	1	453.02	226.51
	29.38	1	28.38	14.19
	29.38	1	28.38	14.19
	54.41	1	53.41	26.705
	35.61	1	34.61	17.305
Average	**87.19916667**	**1.166666667**	**86.0325**	**43.01625**



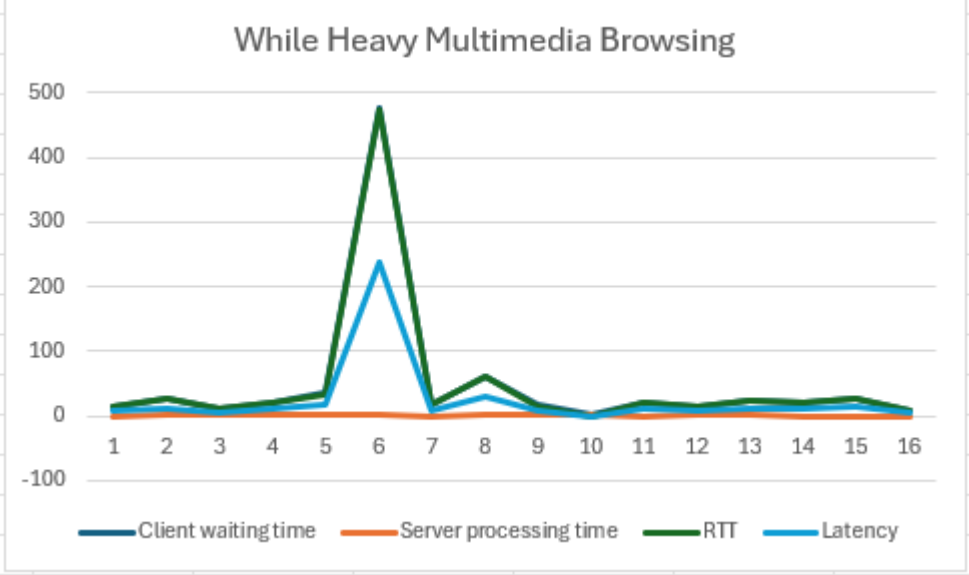
We got a higher average on all calculations except server processing time. We can see two extremely high spikes in delay, which could be due to the streaming of the 4k video from YouTube, which takes a significant amount of bandwidth from the local router. However, after those spikes, the measures went back to their usual values. The explanation for this phenomenon is like the one in the best-case scenario, once the router had successfully found the route to get the YouTube video, it was easier to send requests and receive packages. Additionally, other devices inside the local network might have stopped requiring bandwidth while the test, which releases bandwidth the router can use to efficiently stream video on both the local network and from YouTube.

While Web Browsing

For this test, we deleted all cache and cookies from the server's web browser. After beginning to run the video streaming system, we immediately opened four different web pages with excessive amounts of multimedia content (Instagram, CNN, YouTube, and Facebook) and began to randomly navigate through them to make more requests and saturate the network.

	Client waiting time	Server processing time	RTT	Latency
	17.17	2	15.17	7.585
	23.94	2	21.94	10.97
	12.78	1	11.78	5.89
	41.24	1	40.24	20.12
	27.58	1	26.58	13.29
	16.9	0	16.9	8.45
	118.88	1	117.88	58.94
	113.87	2	111.87	55.935
	31.65	1	30.65	15.325
	245.98	1	244.98	122.49
	10.78	1	9.78	4.89

	20.81	1	19.81	9.905
	129.05	1	128.05	64.025
	14.89	1	13.89	6.945
	25.31	1	24.31	12.155
	26.83	0	26.83	13.415
Average	**54.85375**	**1.0625**	**53.79125**	**26.895625**

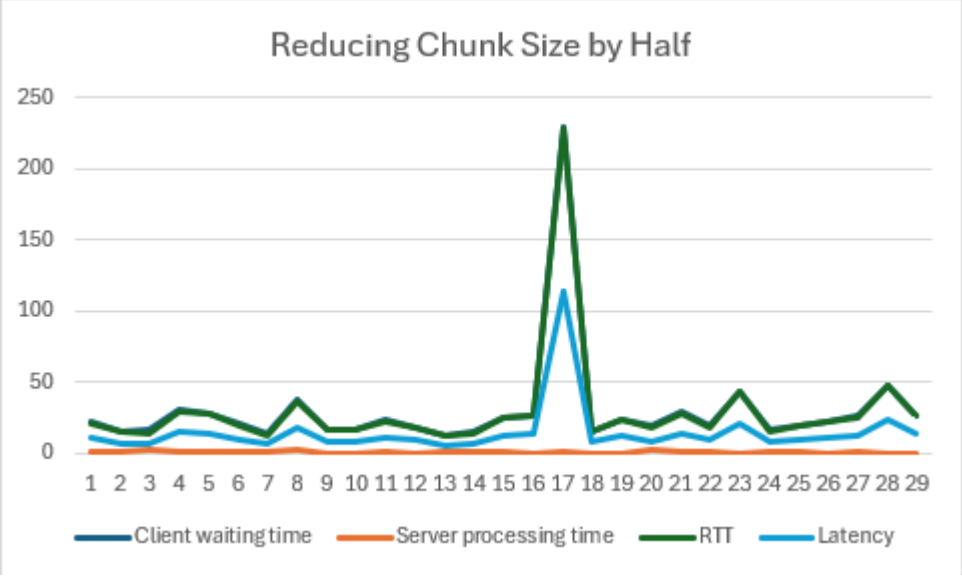


We got a smaller latency average compared to the 4k streaming test, with only one spike in network delays. The spike could be attributed to the many routings and request the network had to sent before being able to address the LAN video streamer. But, similar to the other examples, after the spike the metrics returned to their normal status, but with some smaller spikes that the graph wasn't able to demonstrate due to high difference between one spike on the other ones (towards the end, latency rose to 64 ms before returning to 9 ms). In general, the network presented a higher latency, but this did not affect the client's ability to watch the video without any delays.

Reducing Chunk Size by Half

	Client waiting time	Server processing time	RTT	Latency
	21.52	1	20.52	10.26
	15.4	1	14.4	7.2
	16.08	2	14.08	7.04
	30.39	1	29.39	14.695
	28.42	1	27.42	13.71
	20.18	1	19.18	9.59
	13.27	1	12.27	6.135
	37.62	2	35.62	17.81
	16.25	0	16.25	8.125
	17.06	0	17.06	8.53

	22.83	1	21.83	10.915
	17.74	0	17.74	8.87
	12.66	1	11.66	5.83
	14.49	1	13.49	6.745
	25.49	1	24.49	12.245
	26.89	0	26.89	13.445
	229.62	1	228.62	114.31
	15.15	0	15.15	7.575
	24.15	0	24.15	12.075
	19.23	2	17.23	8.615
	28.98	1	27.98	13.99
	18.88	1	17.88	8.94
	42.65	0	42.65	21.325
	15.91	1	14.91	7.455
	19.74	1	18.74	9.37
	21.82	0	21.82	10.91
	26	1	25	12.5
	47.28	0	47.28	23.64
	25.93	0	25.93	12.965
Average	30.0562069	0.75862069	29.2975862	14.6487931

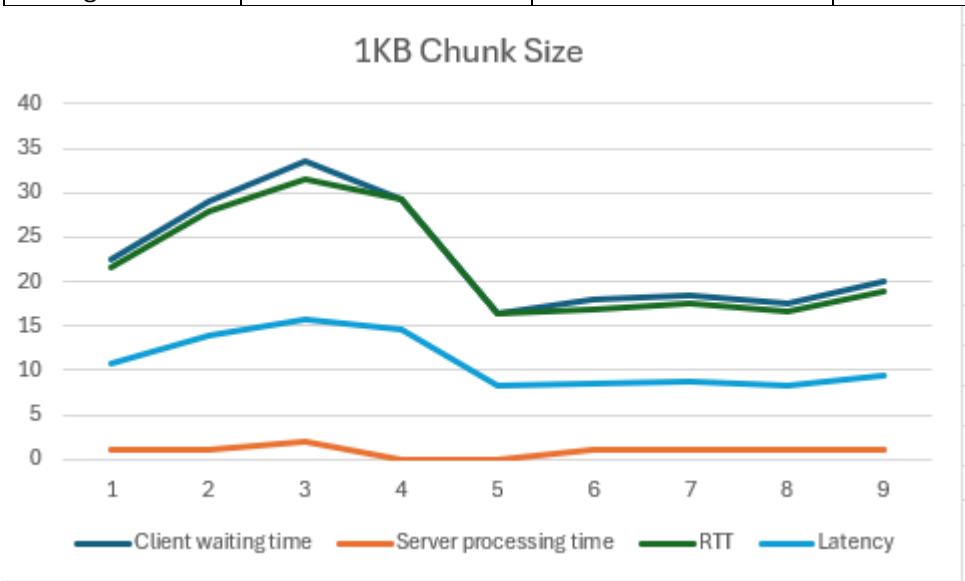


Reducing the chunk size by half (from 1MB to 500000 bytes) did not present much challenge to the network. Regardless of one Spike in latency (lower than the previous spikes), the metrics remained the same and on average, latency was 14 ms, only 4 ms more than the base case. Reducing the chunk size to half of 1MB did not alter the client's viewing experience. It created more entries for the metrics which is obviously since by reducing the size of packages, more packages will be needed for the same video length.

Reducing Chunk Size to 1KB

Since the Chunk Size was reduced by 1000%, the amount of packages is so high that it cannot be shown on this page, so the results are only a sample of the complete set.

	Client waiting time	Server processing time	RTT	Latency
	22.53	1	21.53	10.765
	29	1	28	14
	33.61	2	31.61	15.805
	29.3	0	29.3	14.65
	16.46	0	16.46	8.23
	17.94	1	16.94	8.47
	18.51	1	17.51	8.755
	17.52	1	16.52	8.26
	19.98	1	18.98	9.49
Average	22.76111111	0.888888889	21.8722222	10.9361111



We can see that chunk size did not have a significant effect on the latency of the network, nor the server processing time. Packages where sent and received at similar rates to the base case. However, at the client's end, the video suffered severe delays, making it almost unplayable. The video in total weight 15MB ($15*10^6$ bytes) and lasts 35 seconds, this means that every second of the video weights around 428.57 MB. Following those calculations, we can conclude that 1KB is equal to only 2.3 ms of video. Since the client on average had to wait 22.76 ms for every package, this means that the system needed 22.76 ms to load up only 2.3 ms of video, which resulted in the video having to constantly buffer.

Future Research

A big challenge during this project is not having enough resources to saturate the network. In a future test, we could access the server through multiple devices, and all make requests to the network, which could cause a greater saturation than what we evaluated on this project. Additionally, in a future test we could also use the results from this project to create a Video Streaming service that dynamically decides the Chunk size of the packages based on the server latency calculated.

Conclusions

After analyzing the different test results, we can both confirm and reject certain aspects of the hypothesis. We can first confirm that, due to the Tests being run in the Local Network, the latency was small. Even in the worst case (while streaming 4k YouTube videos) the average latency was 40 ms, which would be considered a good latency for Wide Area Networks. In contrast, it was rejected the hypothesis that web browsing had a significant effect on the network. On the web browsing test, we could see some spikes in latency, but the network came back to its regular latencies after quickly managing that saturation. Another hypothesis that was rejected was that reducing packet size will affect the network. It was demonstrated that small packet sizes did not affect the network since latency remained in the 10-15 ms range in both the half a megabyte and 1 KB test cases. However, having a small packet will have a significant effect on the client's end. A packet smaller than the latency makes the video streaming must constantly buffer the video since it must make and receive requests for lesser amounts of the video.

References

AbdisalanCodes. (2020, 24 octubre). How To Code A Video Streaming Server in NodeJS [Vídeo]. YouTube. https://www.youtube.com/watch?v=ZjBLbXUuyWg