

Agathe Benichou and Frankie Klerer
CS305: Computer Networks
Professor Sadovnik
Due: February 20th 2017

Project 1 - Web Server and Client

Introduction

The goal of this project was to build a simple web application. The program simulates the client- server model used by computer networks. The connection commences with the client establishing a TCP connection with the server via the Internet. The server creates a connection with the client and the client starts sending requests for web pages to the server. The connection is terminated once the server has fulfilled the client's request. The program contains all layers of the Web: application layer (client and server), transport layer, network layer, link layer and physical layer. The client's request is passed down these layers, over to the bottom of the servers layers and back up the servers layers.

HTTP Protocol

After the TCP connection is created between the server and the client, we can start the HTTP protocol. Each of the fields in the HTTP request and response messages contains information pertaining to the page being requested. The client first sends a GET request containing the filename and the version of HTTP as well as other information to give the server information, shown in Figure 1.

```
GET page.html HTTP/1.1  
Host: www.aggieandfrankie.com  
Connection: close  
User-agent: Mozilla/5.0  
Accept-language: eng
```

Figure 1: HTTP GET request

The server receives the request, fulfills the request and transmits a response message back to the client. This response message will depend on the outcome of the request but it always contains a code number to indicate the status on the file requested. For example, 200 is the status code for a successful request, 404 is the status code for the file not being found for an unsuccessful request and 304 is the status code to tell the client that the file hasn't been modified since their last query.

If the request is successful, the response message will contain the date the file was accessed, the date the file was last modified, the file data as well as other important information such as the server and whether the connection is closed or not, shown in Figure 2.

```
HTTP/1.1 200 OK
Date: Sun Feb 19 11:19:57 EST 2017
Server: Apache(Aggie and Frankie)
Last Modified: Sat Feb 18 16:37:02 EST 2017
ETag: 0-23-4024c3a5
Accept-Range: bytes
Content-Length: 171
Connection: close
Content: page.html

Hello!You have downloaded the webpage. **object.html
Congrats! This is the object.
data integrity confirmed
```

Figure 2: HTTP OK response

If there is an embedded object within the data, the client will send another GET request for that object. This GET request is identical to the web pages request but it is requesting the object, rather than web page. Also, the client will have to ask the server to open another connection if HTTP version is non-persistent. This means going through the 3-way handshaking protocol again which delays the fetch time. If the client asks for the same web page again, the GET request message is slightly different. An If-Modified-Since is added to the header of the request message that the client sends, shown in Figure 3a. If the page has not been modified since the date of the last GET request, the server just replies with a Not Modified response message, shown in Figure 3b. Finally, if the web page that the client is looking for cannot be found, then the server will reply with a 404 Not Found status code, shown in Figure 3c.

```
GET page.html HTTP/1.1
If-Modified-Since: Sun Feb 19 11:46:40 EST 2017
Host: www.aggieandfrankie.com
Connection: close
User-agent: Mozilla/5.0
Accept-language: eng
```

Figure 3a: Client GET request with If-Modified Since statement

```
HTTP/1.1 304 Not Modified
```

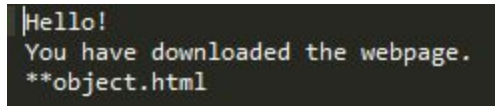
Figure 3b: Server 304 response to clients request

```
HTTP/1.1 404 Not Found
```

Figure 3c: Server 404 Not Found response to clients request

Markup Language

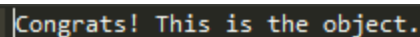
The program needed to read files in the directory whose name was input as a parameter by the client application. In our project, the client is trying to fetch a web page with the name “page.html”. While this file has a .html extension, it is not written in HTML. Figure 4 shows the contents of this page. The last line in the file contains the embedded object written as “**object.html”.



```
Hello!  
You have downloaded the webpage.  
**object.html
```

Figure 4

The client sends a HTTP GET request for the page.html to the server and if all goes well, the server responds with an HTTP OK response as well as with the data from the page.html. The two asterisks at the beginning of the last line tell the client that there is an embedded object in the file and that it needs to fetch that object.html file from the server. Depending on whether the connection is persistent or nonpersistent, the client will either send another HTTP GET request for the object.html or the client will have to establish another connection with the server via a 3-way handshake and then send another HTTP GET request. The server fetches the file from the projects directory, reads in its data (shown in Figure 5) and sends a HTTP OK response with the data of the file.



```
Congrats! This is the object.
```

Figure 5

The client application inserts the object.html data to the page.html data where the object was embedded and displays the final webpage.

Code Design

The client-server model is shown in Figure 6: the HTTP GET request for the web page comes from the client's application layer (indicated as the red square), is sent down through all the clients layers, then sent back up to the servers layers until it reaches the servers application layer. The server fetches the data from the web page and sends it back to the client's application layer. The semantics of how the data is enveloped in a packet with destination headers, pushed through the end systems socket and processed in the layers is not important for this project. However, the program does its best to simulate this.

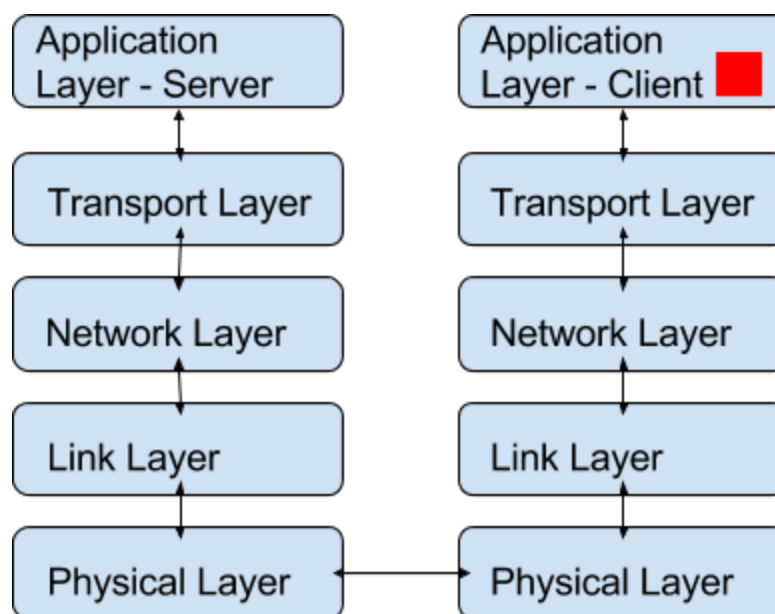


Figure 6: Overview of how the classes relate

In the program, these are the following classes: Server App, Client App, Transport Layer, Network Layer, Link Layer, Physical Layer. The Server and Client App classes are created separately but within they both create the Transport Layer class which creates the Network Layer class which creates the Link Layer class which creates the Physical Layer class. This means that all classes below the Application Layer are created twice so that the Client and Server App classes have their own stack of layers. These layers work together to send messages between end systems. The Application Layer (Client and Server) defines the types of messages being exchanged(request, response), the message syntax and define the protocol for when/how processes send and respond to messages. The Transport Layer processes data transfer and handles TCP/IP data transfer protocols. The Network Layer handles the routing of datagrams between routers. The Link Layer handles data transfer between neighboring network elements. Finally, the Physical Layer propagates the bits of the message between the sender and receiver.

The server app runs on one terminal window with 3 parameters: the transmission delay, the propagation delay and the HTTP version. A sample of the Java terminal line is shown in Figure 7.a. The client app runs on another terminal window with 4 parameters: the transmission delay, the propagation delay, the HTTP version and the web page that is to be fetched. A sample of the Java terminal line is shown in Figure 7.b. The transmission delay, propagation delay and HTTP version must be consistent within the server and client.

```
java ServerApp 0.5 1 1.1
```

Figure 7.a

```
java ClientApp 0.5 1 1.1 page.html
```

Figure 7.b

The server is created first: within the Server App class, the parameters are initialized and the server waits for the Client App class to initiate a TCP connection. Once the Client App class is created, its parameters are also initialized and the client initiates a TCP connection. The TCP connection is a 3-way handshake in which the client sends the server a SYN message to ask if it is ready to create a connection, the server replies with an ACK message to acknowledge the client's SYN message and it creates the TCP connection. Finally, the client starts sending the server requests for web pages. This 3-way handshake is implemented in the Transport Layer class because that is the layer of the Internet that handles data transfer from end systems. TCP is the reliable protocol for transferring data and it guarantees that messages will be delivered between end systems. This interaction is shown in Figure 8. For demonstration purposes, the user must hit Enter on the Client App to continue the fetch.

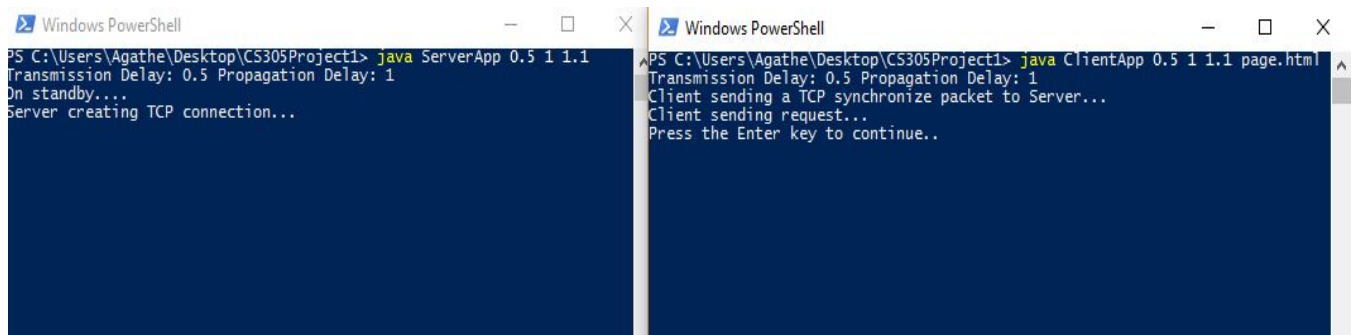


Figure 8: Client Server initializing TCP connection

The Client App class generates a HTTP GET request and sends it to the transport layer which sends it to the network layer. Within the network layer, the transmission delay and propagation delays are used. Every byte of the GET request must be loaded onto the link so for every byte of the GET request, the program is delayed by the input transmission delay. We use an array to keep track of all the bytes and how many have been loaded so far. Once all the bytes are loaded

onto the link, the propagation delay is used to “propagate” the bytes across the link to the server side. This process is shown in Figure 9.

```
PS C:\Users\Agathe\Desktop\CS305Project1> java ClientApp
Transmission Delay: 0.5 Propagation Delay: 1
Client sending a TCP synchronize packet to Server...
Client sending request...
Press the Enter key to continue..

GET page.html HTTP/1.1
Host: www.aggieandfrankie.com
User-Agent: Mozilla/5.0
Accept-Language: eng
Content-Length: 9

Transmitting byte 1 out of 9
Transmitting byte 2 out of 9
Transmitting byte 3 out of 9
Transmitting byte 4 out of 9
Transmitting byte 5 out of 9
Transmitting byte 6 out of 9
Transmitting byte 7 out of 9
Transmitting byte 8 out of 9
Transmitting byte 9 out of 9
Propagating the message across the link...
```

Figure 9: GET request being transmitted onto and across the link

The server receives the bytes and assembles them to rebuild the GET request. The Server App class evaluates whether the file has already been fetched or not and if the file is an embedded object or not. If the file hasn't been fetched and if the file is in the project directory, then the server will read in the file's data and stores the date/time that this file was last modified. Once this is finished, the server sends a HTTP 200 OK response. This response must be transmitted onto the link and propagated across the link back to the client using the same transmission and propagation delays and is shown on the server side in Figure 10.

```
Transmitting byte 78 out of 97
Transmitting byte 79 out of 97
Transmitting byte 80 out of 97
Transmitting byte 81 out of 97
Transmitting byte 82 out of 97
Transmitting byte 83 out of 97
Transmitting byte 84 out of 97
Transmitting byte 85 out of 97
Transmitting byte 86 out of 97
Transmitting byte 87 out of 97
Transmitting byte 88 out of 97
Transmitting byte 89 out of 97
Transmitting byte 90 out of 97
Transmitting byte 91 out of 97
Transmitting byte 92 out of 97
Transmitting byte 93 out of 97
Transmitting byte 94 out of 97
Transmitting byte 95 out of 97
Transmitting byte 96 out of 97
Transmitting byte 97 out of 97
Propagating the message across the link...
On standby..
```

Figure 10: Server transmitting and propagating response back to client

Once the Client App receives this response, it analyzes the data to see if there is an embedded object. If the file's content is found to contain an embedded object, then the client app sends another HTTP GET request for the object's data. If the connection is persistent, the client app will send another GET request because the connection is still open. However, if the connection is non-persistent then the connection closed after the server sent over its response. The client must resend a TCP connection request before sending the GET request for the object. In this case, the connection is persistent so the client just sends another GET request, shown in Figure 11.

```
Receiving the message from the link...
Fetching the embedded object...
GET object.html HTTP/1.1
Host: www.aggieandfrankie.com
User-Agent: Mozilla/5.0
Accept-Language: eng
Content-Length: 9

Transmitting byte 1 out of 13
Transmitting byte 2 out of 13
Transmitting byte 3 out of 13
Transmitting byte 4 out of 13
Transmitting byte 5 out of 13
Transmitting byte 6 out of 13
Transmitting byte 7 out of 13
Transmitting byte 8 out of 13
Transmitting byte 9 out of 13
Transmitting byte 10 out of 13
Transmitting byte 11 out of 13
Transmitting byte 12 out of 13
Transmitting byte 13 out of 13
Propagating the message across the link...
```

Figure 11: Client sending another GET request for embedded object

The Server App repeats its process of fetching the file from the project's directory and sends a HTTP OK response back to the Client app. This response includes the 200 OK response code, the data the file was last modified and the file data as well as other lines. This response is retransmitted onto the link and propagated to the Client app to be displayed there, shown in Figure 12.

```
Transmitting byte 52 out of 74
Transmitting byte 53 out of 74
Transmitting byte 54 out of 74
Transmitting byte 55 out of 74
Transmitting byte 56 out of 74
Transmitting byte 57 out of 74
Transmitting byte 58 out of 74
Transmitting byte 59 out of 74
Transmitting byte 60 out of 74
Transmitting byte 61 out of 74
Transmitting byte 62 out of 74
Transmitting byte 63 out of 74
Transmitting byte 64 out of 74
Transmitting byte 65 out of 74
Transmitting byte 66 out of 74
Transmitting byte 67 out of 74
Transmitting byte 68 out of 74
Transmitting byte 69 out of 74
Transmitting byte 70 out of 74
Transmitting byte 71 out of 74
Transmitting byte 72 out of 74
Transmitting byte 73 out of 74
Transmitting byte 74 out of 74
Propagating the message across the link...
On standby..

Transmitting byte 6 out of 13
Transmitting byte 7 out of 13
Transmitting byte 8 out of 13
Transmitting byte 9 out of 13
Transmitting byte 10 out of 13
Transmitting byte 11 out of 13
Transmitting byte 12 out of 13
Transmitting byte 13 out of 13
Propagating the message across the link...
Receiving the message from the link...

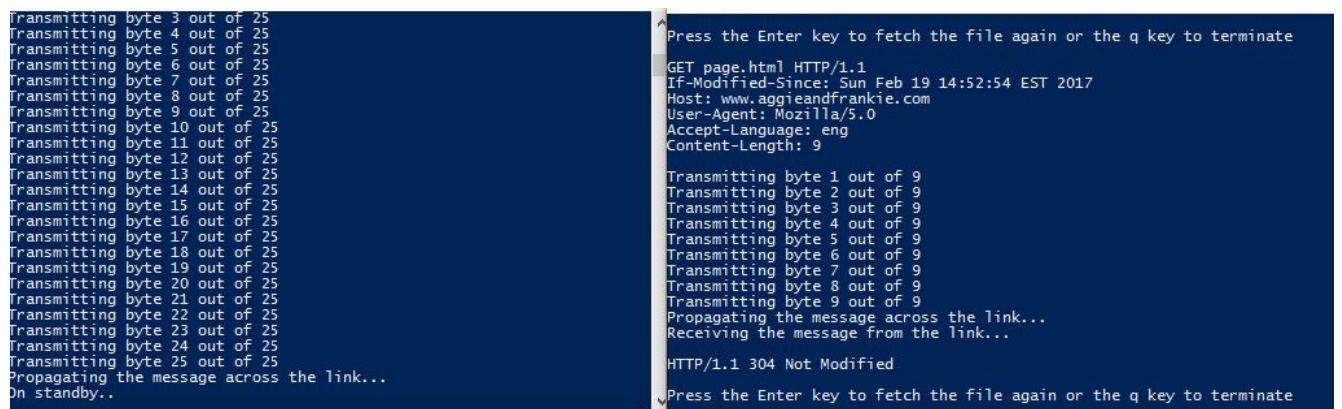
HTTP/1.1 200 OK
Date: Sun Feb 19 14:52:54 EST 2017
Server: Apache(Aggie and Frankie)
Last Modified: Sat Feb 18 16:37:01 EST 2017
ETag: 0-23-4024c3a5
Accept-Range: bytes
Content-Length: 171
Connection: close
Content: page.html

Hello!You have downloaded the webpage. **object.html
Congrats! This is the object.

Press the Enter key to fetch the file again or the q key to terminate
```

Figure 12: Server OK response code transmitted/propagated to client

Once the file and its embedded object has been fetched by the Client App, the user has a choice of refetching the file or quitting the program and receiving the total time it took to fetch the file. If the user decides to refetch the program, the client will send GET request with an if-modified-since statement so that the server knows to only resend the data if the file has been modified since the last time it was accessed by the client. If the file has been modified, then the server will refetch the data from the file and everything will repeat. If the file has not been modified, the server will send a HTTP 304 Not Modified response. This is shown in Figure 13.



```
Transmitting byte 3 out of 25
Transmitting byte 4 out of 25
Transmitting byte 5 out of 25
Transmitting byte 6 out of 25
Transmitting byte 7 out of 25
Transmitting byte 8 out of 25
Transmitting byte 9 out of 25
Transmitting byte 10 out of 25
Transmitting byte 11 out of 25
Transmitting byte 12 out of 25
Transmitting byte 13 out of 25
Transmitting byte 14 out of 25
Transmitting byte 15 out of 25
Transmitting byte 16 out of 25
Transmitting byte 17 out of 25
Transmitting byte 18 out of 25
Transmitting byte 19 out of 25
Transmitting byte 20 out of 25
Transmitting byte 21 out of 25
Transmitting byte 22 out of 25
Transmitting byte 23 out of 25
Transmitting byte 24 out of 25
Transmitting byte 25 out of 25
Propagating the message across the link...
On standby..

Press the Enter key to fetch the file again or the q key to terminate
GET page.html HTTP/1.1
If-Modified-Since: Sun Feb 19 14:52:54 EST 2017
Host: www.aggieandfrankie.com
User-Agent: Mozilla/5.0
Accept-Language: eng
Content-Length: 9

Transmitting byte 1 out of 9
Transmitting byte 2 out of 9
Transmitting byte 3 out of 9
Transmitting byte 4 out of 9
Transmitting byte 5 out of 9
Transmitting byte 6 out of 9
Transmitting byte 7 out of 9
Transmitting byte 8 out of 9
Transmitting byte 9 out of 9
Propagating the message across the link...
Receiving the message from the link...

HTTP/1.1 304 Not Modified
Press the Enter key to fetch the file again or the q key to terminate
```

Figure 13: Server 304 Not Modified response code transmitted/propagated to client

The bulk of the program runs out of the while loops in the Server and Client Apps. This is where request and response messages are analyzed and actions are taken if certain conditions are met. The program was built to handle a one file request at a time with or without at most one embedded object in the file. Requests are passed down from the Client App through methods to the Physical Layer where it is transmitted to the Server Apps Physical Layer class and then passed up through methods.

Results

The program can implement persistent and non persistent connection. In persistent connection, the client and server initiate a TCP connection and the connection remains open throughout the duration of the exchange. In the program, the time it takes to simulate the TCP connection is 6 seconds total. The client sends a SYN message to the server (waits two seconds), the server waits two seconds until it receives the SYN message and responds with an ACK message, and then the client waits two seconds until it receives the ACK message and responds with a GET request for the desired webpage. So before any request has been sent, there is already a delay of 6 seconds. The two second wait times were implemented for demonstration purposes but in reality, the time to create a TCP connection is so little. For this purpose, in the analysis part of the report, 6 seconds has been removed from times before they were graphed. Once a message has been sent from the client app, it is turned to bytes and sent to the transport layer and then the network layer. In the network layer, those bytes need to be transmitted onto the link. The transmission delay occurs for every byte in the byte array of the client's request. So if there are 5 bytes and the transmission delay is 0.5 seconds, the total time to transmit the bytes onto the link is 2.5 seconds. The propagation delay occurs once for all the bytes. So if the propagation delay is 2 seconds, then the total time it takes to propagate all the bytes through the link to the server is 2 seconds. In total and with a transmission delay of 0.5 seconds and a propagation delay of 2 seconds, the time it takes for the server to receive the client's request is 10.5 seconds. The server fulfills the request pretty fast but then has to send the response message back to the client. Depending on the response, the size of the response varies. Say the size of the message is also 5 bytes, this response has to be transmitted and propagated back onto the link to get to the client. This is another 4.5 seconds so the total time for the client to initiate a TCP connection, send a request and receive a response is 15 seconds. This is only if the webpage has no embedded object. If it has an embedded object and assuming the sizes are the same, this is another 9 seconds so a total of 24 seconds that it takes to fetch webpage with an embedded object.

In non persistent connection, the TCP connection closes after the server sends the webpage response. This means that the client sends a SYN message to the server, the server waits until it receives the SYN message and responds with an ACK message, and then the client waits until it receives the ACK message and responds with a GET request for the desired webpage. This process takes another 6 seconds. Up until this point, it has taken 15 seconds. If the webpage has an embedded object, this is another 6 seconds to create a TCP connection and another 10.5 seconds for the request and response. In total, if the connection is non persistent and the webpage has an embedded object is 30 seconds. This is because the TCP connection protocol takes 6 seconds.

D_t = transmission delay

D_p = propagation delay

$\#_b$ = number of bytes

$\#_o$ = number of embedded objects within the web page

$$\text{Persistent connection} = 6 + D_t * \#_b + D_p + D_t * \#_b + D_p + (\#_o)(D_t * \#_b + D_p + (D_t * \#_b(D_t) + D_p))$$

TCP Connection

Client transmits and propagates bytes to link

Server transmits and propagates bytes to link

Repeat for every embedded object

$$\text{Non persistent connection} = 6 + D_t * \#_b + D_p + D_t * \#_b + D_p + (6)(\#_o)(D_t * \#_b + D_p + (D_t * \#_b(D_t) + D_p))$$

TCP Connection

Client transmits and propagates bytes to link

Server transmits and propagates bytes to link

Repeat for every embedded object

For a transmission delay of 0.5 seconds, a propagation delay of 1 second, 5 bytes per request and response:

- Persistent connection with no objects = $6 + 0.5*5 + 2 + 0.5*5 + 2 = 15$
- Persistent connection with 1 object = $6 + 9 + 1(9) = 24$
- Persistent connection with >1 object = $15 + 2(9) = >33$
- Non Persistent connection with no objects = $6 + 0.5*5 + 2 + 0.5*5 + 2 = 15$
- Non Persistent connection with 1 object = $6 + 9 + 1(9) + 6 = 30$
- Non Persistent connection with >1 objects = $15 + 2(9) + 6 = >39$

The local web cache would decrease this time. A web cache temporarily stores web documents to reduce the time it takes to fetch web pages. Instead of the client fetching the web page and any embedded objects from the server, it can check if the local web cache has this web page cached in it already. This would delay the time needed for the server to find the web page.

Analysis

We compared persistent and non persistent using 3 different combinations of transmission and propagation delays. The first combination, indicated with the green circle ● in Figure 14, is with a transmission delay of 0.1 seconds and a propagation delay of 0.1 seconds. The total time it took a non persistent connection to fetch the webpage and its embedded object with these parameters varied between 6 seconds and 8 seconds. These times may seem high but it is a short amount of time compared to the other two parameters. The second combination, indicated with a blue circle ●, is with a transmission delay of 0.5 seconds and a propagation delay of 1 second. The total time it took a non persistent connection to fetch the webpage and its embedded object with these parameters varied between 7 and 11 seconds. The third combination, indicated with an orange circle ●, is with a transmission delay of 1 second and a propagation delay of 2 seconds. The total time it took a non persistent connection to fetch the webpage and its embedded object with these parameters varied between 191 and 201 seconds.

For all of these combination, the transmission delay was multiplied by 5 each time. The transmission delay heavily affected the total time. This is because the transmission delay has to occur for every byte in the HTTP request and response messages. These bytes add up quickly when the server response includes the file content. The propagation delay occurs with every message being sent between the server and client and it remains a constant delay despite the number of bytes that are being sent.

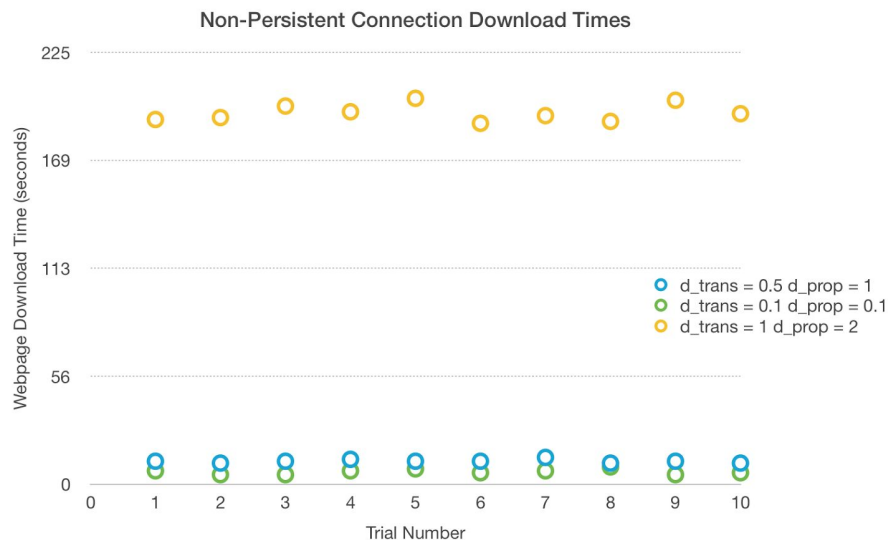


Figure 14: Non-Persistent Connection Download Times Comparison

The combination of times used for persistent connections was the same as the ones used for non persistent connection. The first combination, indicated with the green circle ● in Figure 15, is with a transmission delay of 0.1 seconds and a propagation delay of 0.1 seconds. The total time it took a non persistent connection to fetch the webpage and its embedded object with these parameters varied between 0 seconds and 1 seconds. These times may seem high but it is a short amount of time compared to the other two parameters. The second combination, indicated with a blue circle ●, is with a transmission delay of 0.5 seconds and a propagation delay of 1 second. The total time it took a non persistent connection to fetch the webpage and its embedded object with these parameters varied between 5 and 8 seconds. The third combination, indicated with an orange circle ●, is with a transmission delay of 1 second and a propagation delay of 2 seconds. The total time it took a non persistent connection to fetch the webpage and its embedded object with these parameters varied between 180 and 185 seconds.

The different combinations of transmission and propagation delays varied just as much as they did for non persistent but had lower overall times. This is because there is no need to recreate a TCP connection for fetching the embedded object.

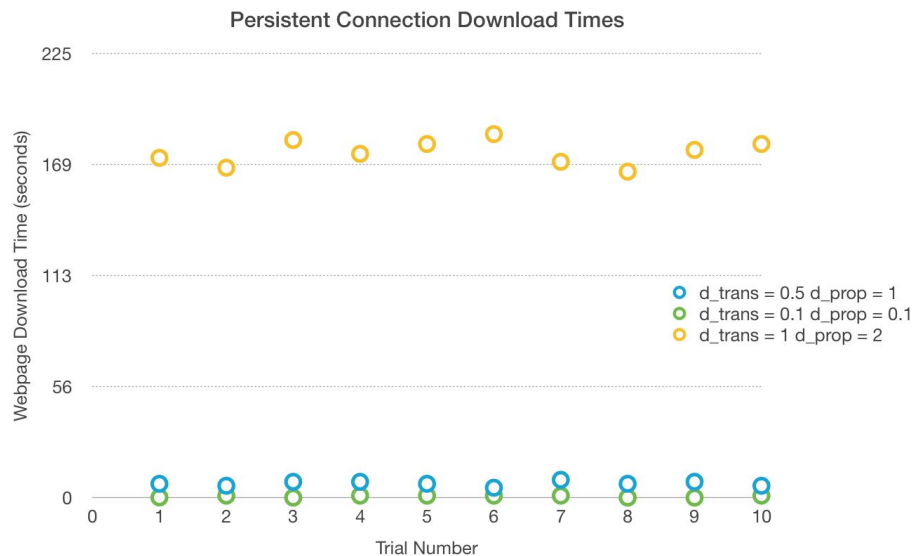


Figure 15: Persistent Connection Download Times Comparison

For the improvements to the HTTP protocol, we looked at Google's SPDY protocol. Google uses four main improvements for dealing with multiple requests. These improvements include prioritized requests, multiplexed requests, and compressing headers to decrease the amount of data transmitted. Since our program is a much smaller version of the overall Internet and we only deal with one request at a time, these improvements would not make a big difference in our program. Only one request is being sent at a time so there is no file request to

prioritize. Instead, we looked into all of the sections on which HTTP protocol can be improved. We decided on implementing error notification and data integrity confirmation messages. We think that it is important for the networks engineer to know what is happening within their program and networks. In our program, if the client and/or server ever run into an error, they will be notified with a detailed error message. In addition, once the page is successfully transmitted and the server confirms that the file has been correctly sent, we can confirm to the user of the data's integrity. Our improvements help the web programs and the programmer find errors in order to easily locate the error.

Conclusion

Under any combination of transmission and propagation delays, the persistent connection has a lower overall time to fetch a website with an embedded object. Non-persistent HTTP connections consistently have slower total transmission time because the TCP connection is closed after the initial web page is received, so the client must re-open the connection using another TCP handshake, slowing down the connection. In terms of parameters, the transmission delay has a larger, variable effect on the overall fetch time, while the propagation delay does not vary much over the conditions since it is consistent. In the real world, transmission and propagation delays are very small and don't have as big of an effect on the total fetch time as they do in in this project. There are many other factors that play into delaying requests and response.

Team

As a team, we both worked on the code and report. Both team members made fair and equivalent contributions to the project. Aggie commented the entire code.

References

"Key Differences between HTTP/1.0 and HTTP/1.1." Key Differences between HTTP/1.0 and HTTP/1.1. N.p., n.d. Web. 20 Feb. 2017.

"SPDY: An Experimental Protocol for a Faster Web." *The Chromium Projects*. N.p., n.d. Web. 20 Feb. 2017.

"Overview | SPDY | Google Developers." *Google Developers*. N.p., n.d. Web. 20 Feb. 2017.