Project 1 Report:
Web Server and Client Simulator

Ha Vu & Thanh Vu
Professor Amir Sadovnik
CS 305: Networks
February 17, 2017

I. Introduction

1. Overview of Concepts

Network applications communicate with one another by the sending of packets through five layers of the Internet (application, transport, network, link and physical) [1]. During any communication attempt between two applications, one host, called the client, requests information, and the other host, called the server, services information.

Among the most important goal of network infrastructure is to minimize delays. One type of delay is propagation delay, or the time necessary for a bit to be transmit from one end of the link to the other end of the link. Transmission delay, meanwhile, is the time necessary to load an entire data packet onto the link. Together, these two delays and a few other types of delay from the end-to-end delay of the network.

Using TCP transport protocol, the sending of information between the client and the server is reliable and connection-oriented. The protocol first establishes the connection with a three-way handshake routine, in which the client sends a SYN message, to which the server will reply with an ACK message if the connection could be established.

Once a connection is established by TCP, the application layers could start sending HTTP requests and responses. HTTP protocol is the Web application-layer protocol that defines how clients request Web objects and servers respond appropriately.

2. Goals

In this project, we aim to simulate the client-server communication through the network using a simplified version of HTTP protocol and TCP, and from there investigate the impact of persistent, non-persistent, as well as an improvement called multiplexing connection on the total response time, as well as the effect of cache on the total response time.

a. Non-persistent, persistent and multiplexing

A TCP connection could either be **persistent** or **nonpersistent**. A persistent connection will be kept open after the application-layer request has been responded, while a non-persistent connection will be closed and have to be re-established. A non-persistent connection frees up listening resources of the server, while a persistent connection potentially saves some of the delay of clients and servers handshaking.

An improvement we added to TCP is **multiplexing**. Inspired by SPDY [2], multiplexing allows multiple objects to be requested within one connection, but the connection still closes after all the objects have been received. In other words, multiplexing allows client to request a list of objects with one handshake protocol. This reduces the number of handshakes compared to non-persistent connection, but still frees up resources when not needed, compared to persistent.

We hypothesize that persistent connection will have the shortest delay, then multiplexing, and finally nonpersistent.

b. Cache and non-cache

One way to minimize the end-to-end delay is to implement a **local cache** in the client application. A file requested by clients can be cached by the local browser. If the clients have requested the same object before, the local application may decide to cached it, meaning storing a copy of the object locally. The cache sends an if-modified request to the server to check if the file is up-to-date, and responds to the clients with that copy if this is the case. If it is not, the local cache returns a new copy of the file that it gets from the server. This mechanism allows clients to experience faster response time. Local cache eliminates network delays by minimize the number of requests to the server and reduces the amount of traffic in the network.

We hypothesize that caching will reduce overall delay in getting objects.

3. Contributions

Ha

- Server application
- HTTP packets (request and response)
- Network delays
- Three-way handshakes
- Experiment Controller

Thanh

- Client application
- Local cache
- Persistent and nonpersistent connections
- Document (format of markup language)
- Multiplexing improvement

Both

- Overall design
- Experiment
- Report

II. Design

1. HTTP Protocol

a. Syntax

Request message:

Method	sp	URL	sp	Version	crlf
heading1:	sp	value1	crlf		
heading2:	sp	value2	crlf		
crlf					

Response message:

Version	sp	Status Code	sp	Phrase	crlf
heading1:	sp	value1	crlf		
heading2:	sp	value2	crlf		
crlf					
Body					

b. Semantics

Request message:

- Method: The method that is used to request information
    - GET: request a file with the URL specified
- URL: The path to file.
- Version: The HTTP version.
    - HTTP/1.0 is non-persistent
    - HTTP/1.1 is persistent connection
    - HTTP/1.2 is our multiplexing improvement

- Headers:
    - If-modified-since: Only added if a version is found in the local cache. The value is a timestamp, and this turns the GET request into a conditional GET.
    - Close-after-this: HTTP/1.2 header, which lets server know whether to close connection after sending packet or not.

Response message:

- Version: The HTTP version.
    - HTTP/1.0 is non-persistent
    - HTTP/1.1 is persistent connection
    - HTTP/1.2 is our multiplexing improvement
- Status code and phrase:
    - 200 OK: Displayed when file is found and transferred over.
    - 304 Not Modified: Displayed in response to a conditional GET, only if the cached modified time is the same as the modified time of the version of the server file.
    - 404 Not Found: Displayed when the URL does not return any file.
- Headers:
    - Last-Modified: The value of this header is the timestamp of the file attached
    - URL: The path to the requested file

c. Expected Exchange of Messages

HTTP/1.0 and HTTP/1.1

Process	**Message exchanged**
Handshake	1. Client sends SYN (transport layer) 2. Server sends ACK (transport layer)
Request and response	**If requested object is not cached:** 3. Client sends GET request 4. Server responds: If found on server: 200 OK response and data If not found on server: 404 Not Found **If requested object is cached:** 3. Client sends conditional GET request 4. Server responds: If found on server and file was modified: 200 OK response and data If found on server and file was not modified: 304 Not Modified response If not found on server: 404 Not Found
Close connection	**HTTP/1.0:** Close **HTTP/1.1**: Not close

HTTP/1.2: Multiplexing

Process	**Message exchanged**
Handshake	1. Client sends SYN (transport layer) 2. Server sends ACK (transport layer)
Request and response	**If requested object is not cached:** 3. Client sends a list of GET requests, with a *Close-after-this* header that tells the server to close the connection after handling the last request. 4. Server responds: If found on server: 200 OK response and data If not found on server: 404 Not Found **If requested object is cached:** 3. Client sends conditional GET request, with a *Close-after-this* header that tells the server to close the connection after handling the last request. 4. Server responds: If found on server and file was modified: 200 OK response and data If found on server and file was not modified: 304 Not Modified response If not found on server: 404 Not Found
Close connection	**HTTP/1.2:** Server closes when *Close-after-this* header is *"true"* in the GET message. Client closes when the last one in the list is received.

2. Markup Language

A marked up file contains normal ASCII text with 0, 1, or more references to other markup files. The URLs of reffered elements are surrounded by '<<' and '>>'.

Example: Here pA1.txt has a reference to cA11.txt while cA11.txt has no reference.

pA1.txt

pA1.txt
1 child:
<<school/cA11.txt>>
data data data data
data data data data
data data data data
data data data data
data data data data
data data data data

data data data data
data data data data

home/cA11.txt

Hi.
I am cA11.txt
data data data data data data data data data
data data data data data data data data data
data data data data data data data data data
data data data data data data data data
Bye.

3. Design and Data Structures

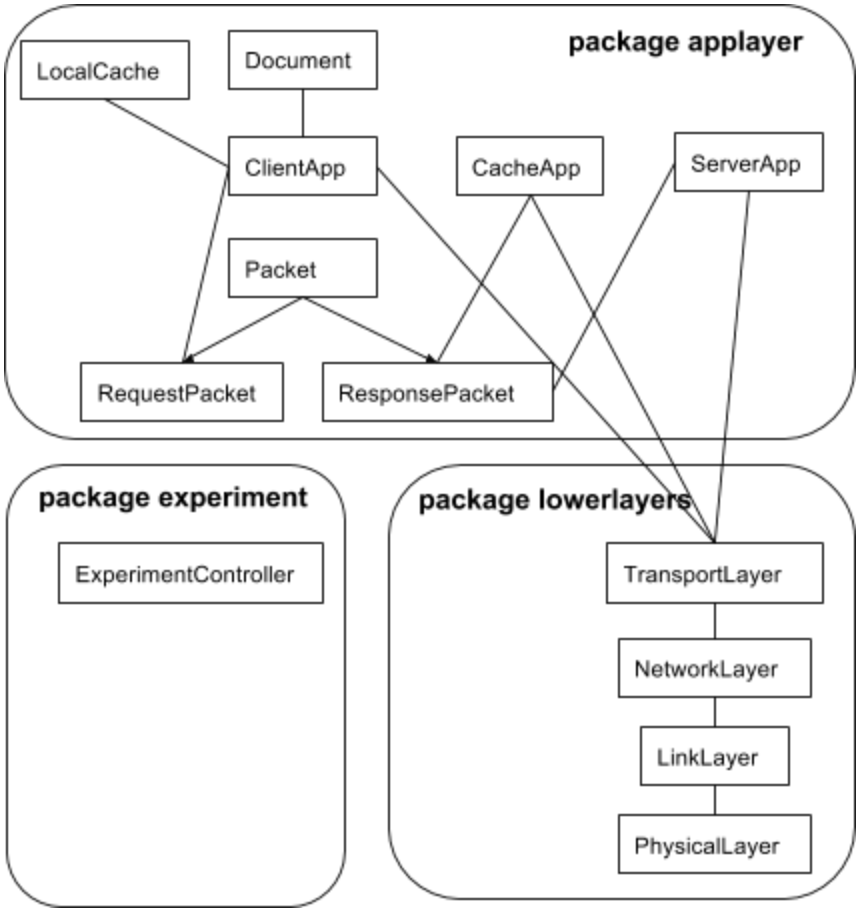


Figure 1: A class diagram of the program

The program is divided into three packages: `applayer`, `lowerlayers` and `experiment`. The `applayer` package includes all the classes in the application layer, including the implementation for both client (`ClientApp`) and server (`ServerApp`), the packets, the cache

and the document display. The `lowerlayers` package consists of the other 4 layers. The `experiment` package consists of `ExperimentController`, which runs all the experiments for the program.

a. Implementation of delays

Transmission delay and propagation delay are implemented in the NetworkLayer class in the method `delay(byte[] payload)`, which is called everytime either the **client** calls `send(byte[] payload)` or `receive()`. Transmission delay varies with the size of payload, while propagation delay is fixed. The client controls these parameters, because `ExperimentController` can only manipulates from the client side.

b. Implementation of handshakes

Handshakes are implemented in `TransportLayer`, accomplished by checking if there's a connection open whenever client or server sends/receives something. If not, the client will need to sends a SYN message and waits for an ACK message, and the reverse for the server, and then a connection will be open for both (represented as a boolean).

c. Implementation of HTTP

HTTP is implemented through the `Packet` class, which consists of 2 children: `RequestPacket` and `ResponsePacket`. Each packet class consists of a `toProtocol()` and a `parseProtocol(String protocol)` method, which translates a packet object to HTTP protocol and vice versa. Each packet object has all the fields as defined above. The headers are implemented as a `HashMap<String, String>`, with the header being the key and the value being the mapping, which ensures O(1) complexity of searching for information within the packet.

d. Algorithm of persistent vs. non-persistent vs. multiplexing

A boolean `connectionOpen` is kept in `TransportLayer`. To close a connection, the boolean is set to false, and a handshake routine will need to be performed to open it again the next time.

Non-persistent (HTTP 1.0):

On client side, the connection is closed after receiving the response packet. On the server side, the connection is closed after sending the response packet.

Persistent (HTTP 1.1):

The connection is never closed.

Multiplexing (HTTP 1.2):

On client side, the client takes a list of GET requests and open one single TCP connection to retrieve all of them. In other words, the handshake happens at the beginning and the connection is kept open while objects in the list are requested one by one. Client closes the connection when all objects are received.

On the server side, the server looks at *Close-after-this* header in the GET request to determine when to close the connection. If the request is the last one in the list, *Close-after-this* will be equal to “true” and the server will close after sending that object.

e. Implementation of caching:

Cache is implemented in class `LocalCache` using `HashMap<String, CachedObject>` as the data structure. The key of the mapping is the cache URL; the value is a custom class that consists of information from that object: the URL, the content and the last modified time.

On the client side, before sending a GET request, the client will check its `LocalCache` to see whether the URL exists in the hashmap. If not found, it continue to send a normal GET request. If found, it retrieves the last modified time and add it to the header of the GET request ("If-modified-since") to form a conditional GET.

On the server side, the server will check the GET request for that header. If found, the server will compare the time to its own object's last modified time, and send back a 304 Not Modified response packet if the modified times are the same. Otherwise, it'll send back a 200 OK response with the object included as usual.

For every 200 OK response the server sends over, the client will update the cache with the latest object, content and last modified time.

f. Algorithm to display embedded documents

Each markup file received from the server is represent as a Document object. Upon requesting a file, the client will recursively retrieve all embedded documents within that parent document. When being displayed, document object will recursively replace its embedded urls with the content of corresponding embedded Document objects, then print out the entire text to the terminal.

III. Results & Analysis

1. Correctness

The raw correctness results could be found in Appendix 1. For all correctness experiments, propagation delay is 200 ms, transmission delay per byte is 2 ms, size of a control packet ranges between 24 - 43 bytes (average 33 bytes), size of a data packet is about 269 bytes. The experiment is run 3 times each, and the averaged is recorded.

Delay to send a control packet = d_{prop} + $d_{transpb}$ * size = 200 + 2*33 = 266 ms

Delay to send a data packet = d_{prop} + $d_{transpb}$ * size = 200 + 2*269 = 738 ms

For each experiment, the controller downloads 2 files with 2 embedded objects each, pA2.txt and pB2.txt (6 files in total).

a. Persistent HTTP

Manual Calculation:

Client and server handshake once, resulting in 2 control packets. Each GET request is a control packet, and there are 6 data packets sent back. Thus, in total, there should be 8 control packets and 6 data packets being sent in total.

Total delay =8 * $d_{control}$ + 6 * d_{data} = 8*266 + 6*738 = 6566 ms.

Actual Result: 6198 ms (correct)

b. Non-persistent HTTP

Manual Calculation:

Client and server handshake six times, once for each packet, resulting in 12 control packets. Each GET request is a control packet, and there are 6 data packets sent back. Thus, in total, there should be 18 control packets and 6 data packets being sent in total.

Total delay = 18 * $d_{control}$ + 6 * d_{data} = 18*266 + 6*738 = 9216 ms

Actual Result: 8572 ms (correct)

c. Cached (non-persistent)

Manual Calculation:

Client and server handshake six times, resulting in 12 control packets. Each GET request is a control packet, and the responses are also control packets since the files are cached. Thus, in total, there should be 18 control packets being sent in total.

Total delay = 24 * $d_{control}$ = 24*266 = 6384 ms

Actual Result: 6311 ms (correct)

d. Multiplexing HTTP

Manual Calculation:

Client and server handshake three times, the first for 2 parents, and the others for the children of respective parent files, resulting in 6 control packets. Each GET request is a control packet, and there are 6 data packets sent back. In total, there should be 12 control packets and 6 data packets in total.

Total delay = 12 * $d_{control}$ + 6 * d_{data} = 12*266 + 6*738 = 7620 ms

Actual Result: 7458 ms (correct)

2. Experiment and Result Analysis

We measure response time against three independent variables: transmission delay per byte, propagation delay and number of objects sent (see Appendix 2). Each graph has 10 data points, and each data point is the average of 3 trials. For all of these three experiments, we measure the response times for non-persistent, persistent and multiplexing connection. For transmission delay experiment, we also measure the response time for retrieving cached objects with a non-persistent connection.

# of parent files	= P
# of child files (per parent)	= C
Size of control packet	= S_{ctrl}
Size of data packet	= S_{data}
Delay to send control packet	= d_{prop} + d_{trans} * S_{ctrl} = d_{ctrl}
Delay to send data packet	= d_{prop} + d_{trans} * S_{data} = d_{data}

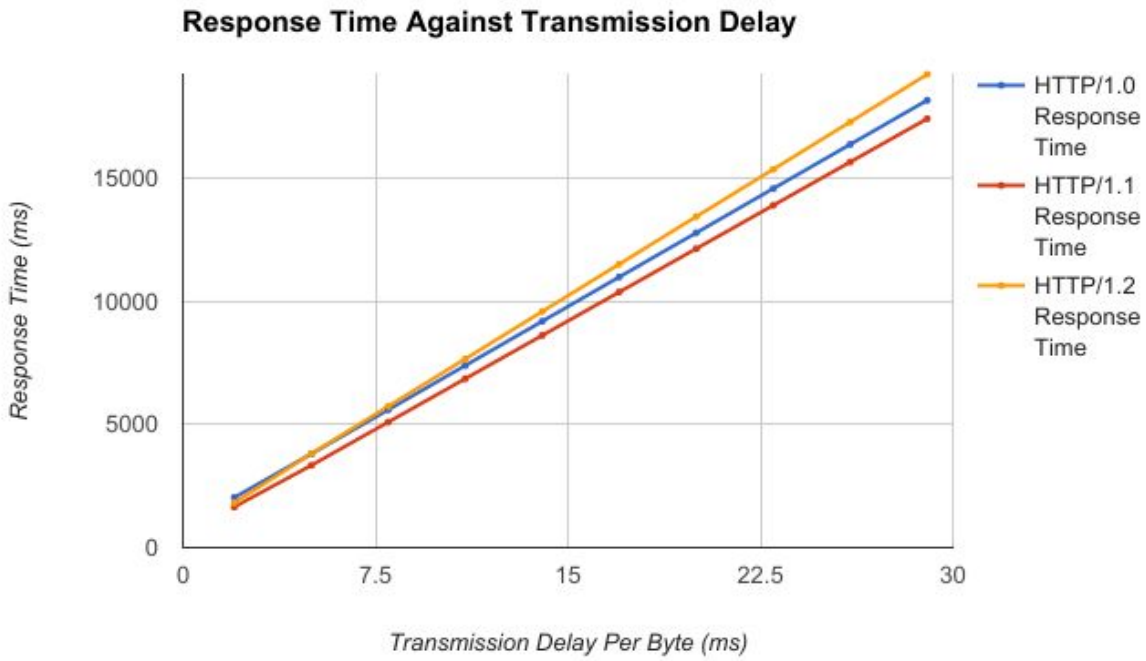
Expected persistent connection delay
= $2*d_{ctrl} + (P + C*P) * (d_{ctrl} + d_{data})$ = $(2 + P + P*C) *d_{ctrl} + (P + C*P) *d_{data}$
Expected non-Persistent connection delay
= $(P + C*P) * (3*d_{ctrl} + d_{data})$ = $3*(P + C*P) *d_{ctrl} + (P + C*P) *d_{data}$
Expected multiplexing connection delay
= $(3*d_{ctrl} + d_{data}) *P + (2*d_{ctrl}*P) + (d_{ctrl} + d_{data})*P*C$ = $(5*P + P*C) *d_{ctrl} + (P + C*P) *d_{data}$

a. Transmission delay

Parameters:
Control Variables:
Number of objects requested: 2
Propagation Delay: 100 ms

Independent Variable:
Minimum Transmission Delay Per Byte: 2 ms
Maximum Transmission Delay Per Byte: 29 ms
Increment: 3 ms

Results:



The response time correlates linearly with the transmission delay per byte for all three connections. Overall, multiplexing (HTTP/1.2) seems to trend towards a higher response time overall, than non-persistent (1.0) and persistent (1.1). However, the difference between the three in terms of response time are not significant.

This result slightly contradicts our hypothesis. That 1.0 outperforms 1.2 is unexpected because 1.2 should have had fewer control packets than 1.0. However, since we only send 2 files, this difference is minimal (2 control packets). Furthermore, 1.2 has to send bigger response packets due to an additional header, and because of our small number of files sent, this differences in packet size probably trumped the differences in number of control packets sent, amplified by a bigger transmission delay per byte.

b. Propagation delay

Parameters:

Control Variables:

Number of objects requested: 2

Transmission Delay per Byte: 2 ms

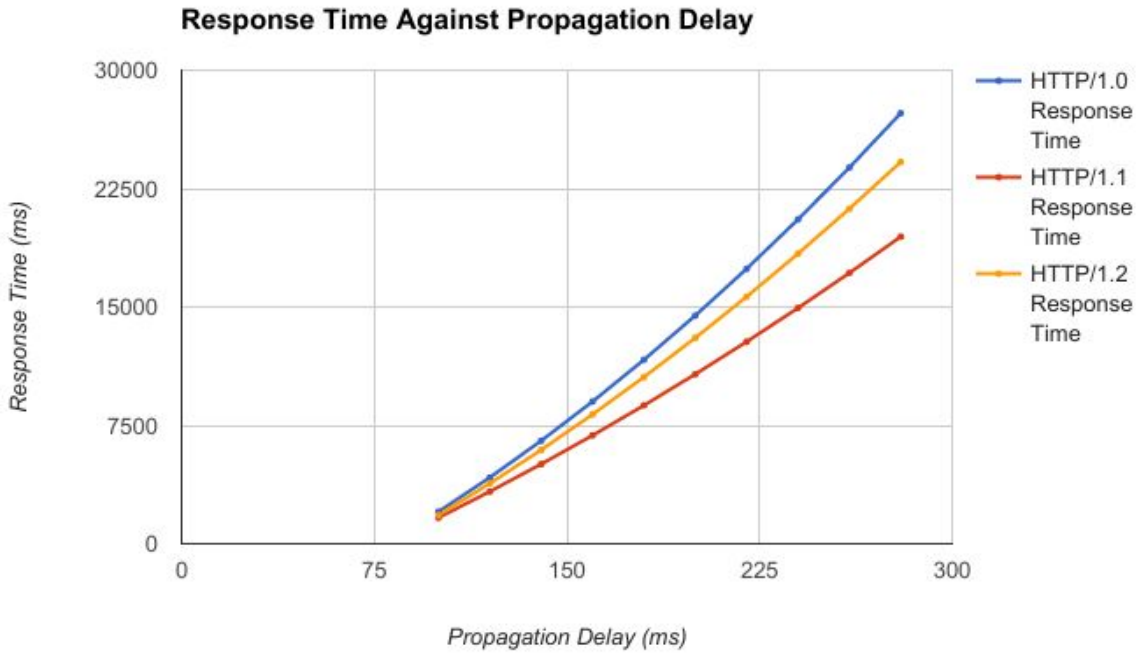
Independent Variable:

Minimum Propagation Delay: 100 ms

Maximum Propagation Delay: 280 ms

Increment: 20 ms

Results:



The response time overall increases as propagation delay increases for all three connections. As this delay increases, the three graphs diverge significantly, with persistent connection being the fastest, then multiplexing, and then non-persistent.

This result supports our hypothesis, as an increase in propagation delay amplifies the impact of sending more total number packets. Persistent connection sends the smallest number of packets, followed by multiplexing and then nonpersistent, hence the divergence in timing.

c. Number of objects

Parameters:

Control Variables:

Transmission Delay per Byte: 2 ms

Propagation Delay: 100 ms

Independent Variable:

Number of parent objects: 2

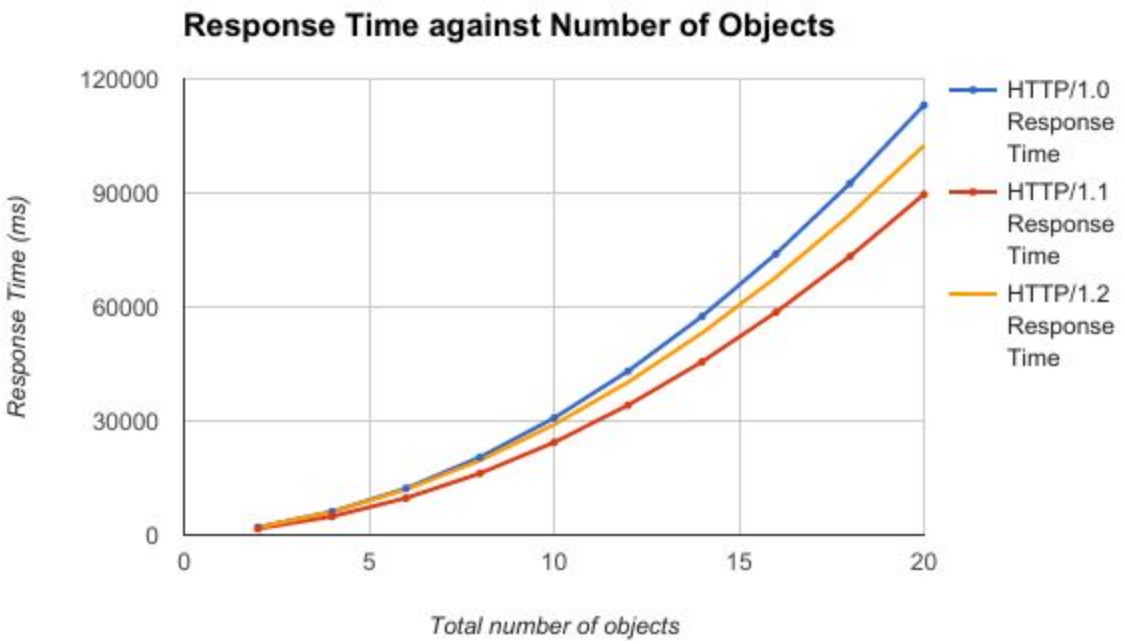
Minimum number of children objects per parent: 0

Maximum number of children objects per parent: 9

Total number of objects requested: 2 - 20

Increment: 2 objects

Results:



The response time overall increases as total number of objects downloaded increases for all three connections. As this number increases, the three graphs diverge significantly, with persistent connection being the fastest, then multiplexing, and then non-persistent.

This result supports our hypothesis. An increase in number of object downloaded increases the number of handshakes that non-persistent connection has to perform, while persistent connection only needs to handshake once. Multiplex in this case maintains the number of handshakes at maximum three times, as all the objects downloaded are either parents or children of existing parents, allowing multiplexing to maintain the connection. The difference in response time reflects the difference in number of control packets due to handshaking between each type of connection.

d. Transmission delay for caching

Parameters:

Control Variables:

Number of objects requested: 2

Propagation Delay: 100 ms

HTTP Version: 1.0

Independent Variable:

Minimum Transmission Delay Per Byte: 2 ms

Maximum Transmission Delay Per Byte: 29 ms

Increment: 3 ms

Results:

The response time correlates linearly with the transmission delay per byte for downloading both cached and non-cached objects. Overall, downloading cached objects significantly outperforms downloading uncached objects in terms of response time, with the two graphs diverging as transmission delay increases.

This supports our hypothesis that caching reduces response time. As the server does not have to send back data to the client when the data is already cached, the total transmission delay is thus lower. An increase in transmission delay per byte thus amplifies the total response time difference, leading to this result.

IV. Conclusion

Overall, persistent connection HTTP/1.1 provides the smallest response time among the three versions, while non-persistent connection HTTP/1.0 has the largest delay. This is generally true regardless of whether it is transmission delay per byte, propagation delay, or number of requested objects that varies. These experiment results agree with our expectations that 1.1 will outperform the other two versions, 1.0 and 1.2, in terms of response time due to its minimum number of TCP handshake packets. However, with persistent connection, the server always needs to keep its connections open even when the client may have already received the

requested objects. Thus, this type of connection is beneficial for the client's experience but costly for the server.

An alternative solution could be to exploit the multiplexing algorithm, HTTP/1.2. According to our experimental data, although the delay is still longer than HTTP/1.1, multiplexing shows an evident improvement over HTTP/1.0. Moreover, since the connection is closed when all objects in the request list are received, multiplexing connection could theoretically help with more efficient allocation of resources on the server side.

Finally, our results also illustrate the clear benefits of using local cache to significantly improve the response time.

V. Limitation and Future Work

With the current input parameters, the results of our transmission delay experiment are inconsistent with our expectation. Due to the time limitation of the project, we unfortunately could not further examine these data. However, we suspect that the reasons for the unexpected behavior may lie in the fact that both the number of requested objects and their sizes are quite small, only 2 objects of 200 bytes each. On one hand, this amplifies the negative effect of additional header tags in HTTP/1.2 requests and responses, which should be negligible for actual web pages. And on the other hand, it reduces the positive effect of having less of control packets, which could be more visible with larger number of requested objects and embedded files per objects.

VI. References

[1] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-down Approach*. Addison-Wesley Longman, 2010.

[2] A. A. Salam, A. Abdel Salam, M. Luglio, C. Roseti, and F. Zampognaro, "SPDY multiplexing approach on long-latency links," in *2014 IEEE Wireless Communications and Networking Conference (WCNC)*, 2014.

Appendix 1: Raw Correctness Checking Results

Correctness for HTTP 1.0 with caching	
HTTP Version	1
Propagation Delay	200
Transmission Delay Per Byte	2
Uncached Response Time	8572
Cached Response Time	6311
Correctness for HTTP 1.1	
HTTP Version	1.1
Propagation Delay	200
Transmission Delay Per Byte	2
Uncached Response Time	6198
Correctness for HTTP 1.2	
HTTP Version	1.2
Propagation Delay	200
Transmission Delay Per Byte	2
Uncached Response Time	7458

Appendix 2: Raw Experiment Results

Transmission Delay vs. Response Time										
HTTP Version	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Transmission Delay Per Byte	2	5	8	11	14	17	20	23	26	29
Uncached Response Time	2030	3800	5592	7386	9180	10977	12770	14564	16357	18147
Cached Response Time	1264	1941	2618	3295	3973	4652	5329	6006	6685	7362
Transmission Delay vs. Response Time										
HTTP Version	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1
Transmission Delay Per Byte	2	5	8	11	14	17	20	23	26	29
Uncached Response Time	1650	3335	5092	6851	8608	10367	12125	13884	15642	17399
Transmission Delay vs. Response Time										
HTTP Version	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2
Transmission Delay Per Byte	2	5	8	11	14	17	20	23	26	29
Uncached Response Time	1815	3808	5732	7654	9578	11500	13426	15346	17269	19198
Propagation Delay vs. Response Time										
HTTP Version	1	1	1	1	1	1	1	1	1	1
Propagation Delay	100	120	140	160	180	200	220	240	260	280
HTTP/1.0 Response Time	2028	4197	6524	9010	11654	14458	17421	20548	23833	27277
Propagation Delay vs. Response Time										
HTTP Version	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1
Propagation Delay	100	120	140	160	180	200	220	240	260	280
Uncached Response Time	1651	3309	5047	6866	8765	10743	12800	14937	17157	19455
Propagation Delay vs. Response Time										
HTTP Version	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2

Propagation Delay	100	120	140	160	180	200	220	240	260	280
Uncached Response Time	1815	3820	5946	8193	10559	13046	15651	18377	21223	24189
Number of Objects vs. Response Time										
HTTP Version	1	1	1	1	1	1	1	1	1	1
Number of parent objects	2	2	2	2	2	2	2	2	2	2
Number of children per parent	0	1	2	3	4	5	6	7	8	9
Total number of objects	2	4	6	8	10	12	14	16	18	20
Uncached Response Time	2045	6116	12257	20473	30758	43105	57513	73985	92516	113124
Number of Objects vs. Response Time										
HTTP Version	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1
Number of parent objects	2	2	2	2	2	2	2	2	2	2
Number of children per parent	0	1	2	3	4	5	6	7	8	9
Total number of objects	2	4	6	8	10	12	14	16	18	20
Uncached Response Time	1646	4855	9713	16209	24351	34141	45559	58613	73305	89640
Number of Objects vs. Response Time										
HTTP Version	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2
Number of parent objects	2	2	2	2	2	2	2	2	2	2
Number of children per parent	0	1	2	3	4	5	6	7	8	9
Total number of objects	2	4	6	8	10	12	14	16	18	20
Uncached Response Time	1979	6084	11970	19615	29036	40194	53126	67831	84285	102498