Erman HAVUÇ
150114023

Sümeyra GÜLSOY
150114028

COMPUTER NETWORKS

HTTP & PROXY SERVER PROJECT REPORT

We developed our http server in python with `socket`, `sys`, `os`, `datetime` and `thread` packages. We create thread for every request. We write `create_headers_errors(status, filetype, size)` method that takes status, filetype, size and create header and error status and if status is 200 return headers otherwise, method returns headers with error status.

`create_response(request)` is method which takes request such that `GET http://localhost:8080/500 HTTP/1.0` and splits the request by spaces and take request code such that GET, POST etc. and URI of request without '/'. After separating the request, Method checks the size of request. Size bounds should be between 100 and 20000 for HTTP server. If size bound is okay, method search the '`ctl`' file to return chai tea latte recipe to client as a response with desired length. After that, method checks the request for errors and call `create_headers_errors()` with error codes.

`server_thread(cc)` is method which take connection client object. Firstly, method takes request of client as 1024 byte. Method calls `reg.decode()` that converts binary code to string and method prints first row of request. After that, Method calls `create_response()` with response and split by '\r\n' and print first row of response .Method converts response from string to binary code with `response.encode()` call and send the response with calling `sendall()` method on connection client object. After that, method unlocked the thread with release call. Finally, method closes the connection.

We write main function to manage the all process. Firstly, method checks the command line argument if user enters special port number to connection, method accepts this port number to connect. If user doesn't enter anything, default port number is `8080` for our http server. We assign host number is `'127.0.0.1'`. After that, we use socket programming with `socket.socket()` method that takes `socket.AF_INET`, `socket.SOCK_STREAM` parameters .These parameters is used for socket type IPv4 option and TCP connection. Another method for socket programming is `setsockopt` that is set the options of socket with reusable port. `socket.bind()` is used for binding socket to host and port. Finally, we used `socket.listen()` to listen requests that comes to socket.

We provide client connection to server with `socket.accept()`. We locked the thread after connection to server. We call `start_new_thread(server_thread, (client_connection,))` from thread package .This method creates new connection with request.

For proxy server implementation, we use `socket`, `hashlib`, `thread` packages. We create thread for every request. We use the same method that is `create_headers_errors(status, filetype, size)`. This method returns different error codes that is only difference from http server.

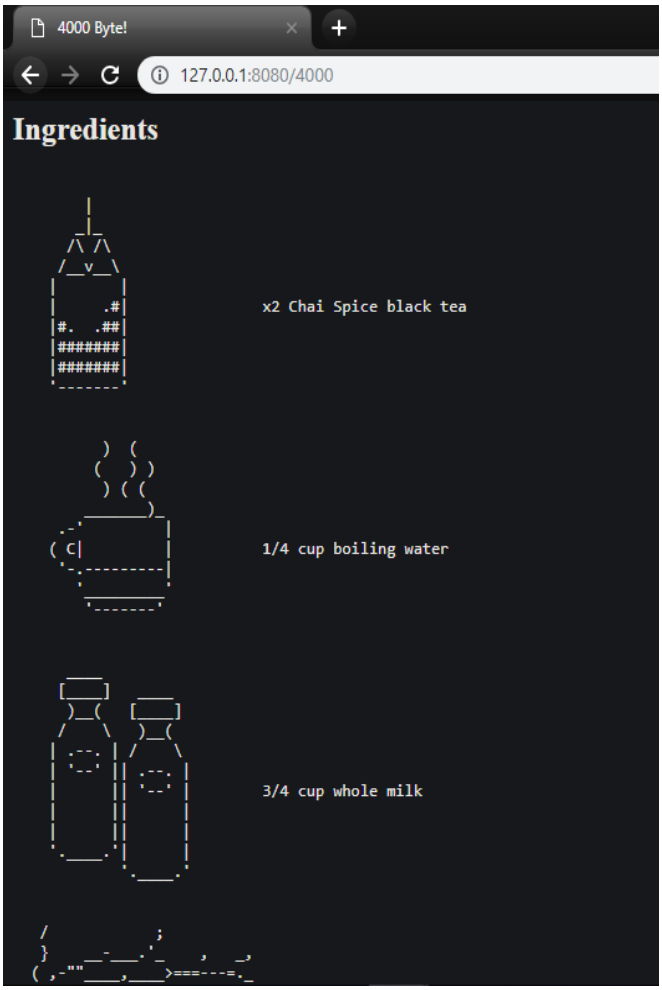
We write `create_response(request)` method again with some differences. In this method, we splits the request by space and we get the request code from split request. If code is GET, methods checks the URI to be digit only. If size is less than 9999, method selects the encryption type according to `hashlib.md5()`. Method encrypts the URI according to size to hold cache name. With this cache name, method checks the existence of this cache in the path. If cache exists, method opens cache file and send the data from cache file. If cache is miss, methods connects the http server and get data from server and write to new cache file and send to client.

All the process is built with `socket.socket(socket.AF_INET, socket.SOCK_STREAM)` , `connect()` and `sendall()` method calls. Here, if request is done from proxy server directly, method redirect the request to http server as default.

We use same `main()` and `server_thread(cc)` logic in Proxy server code.

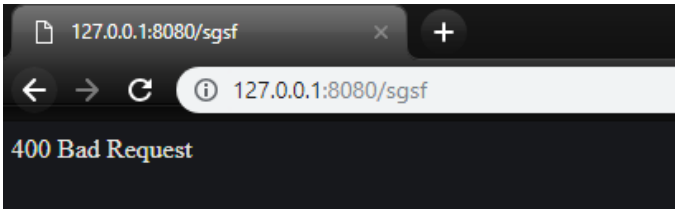
Screenshots from execution process:

1. 200 OK



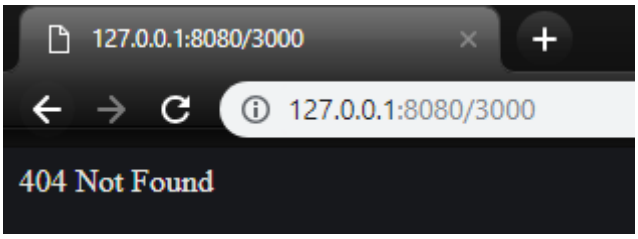
2. 400 Bad Request

We try to enter URI with non-digit.



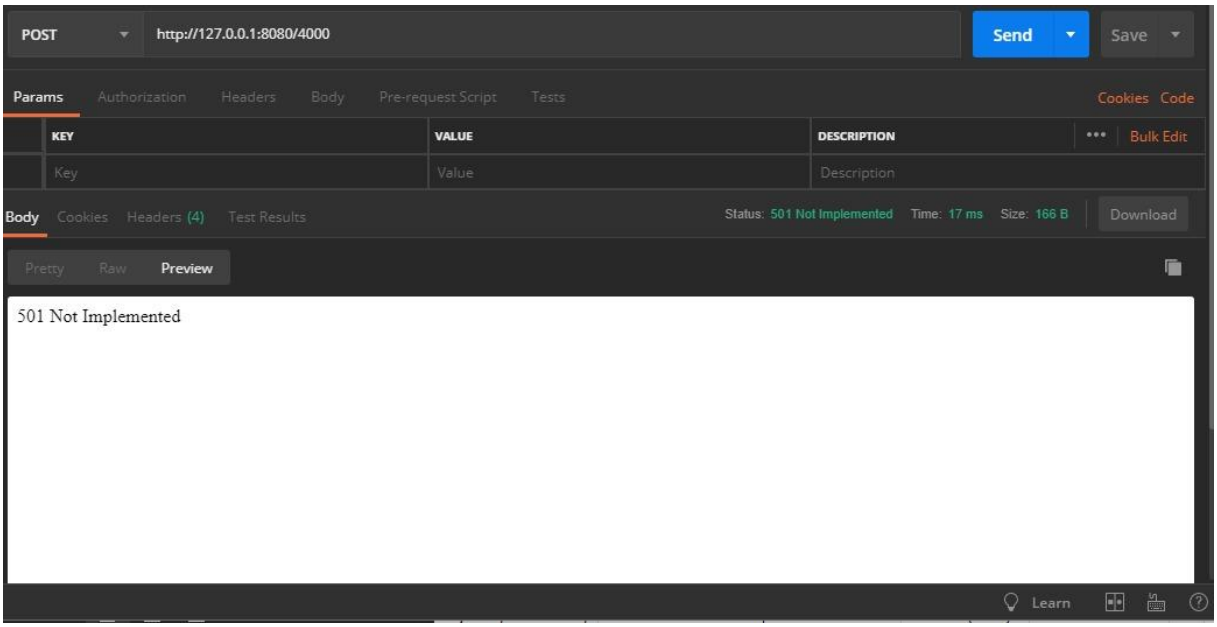
3.404 Not Found

We try to reach http server from Proxy server when http server is closed.



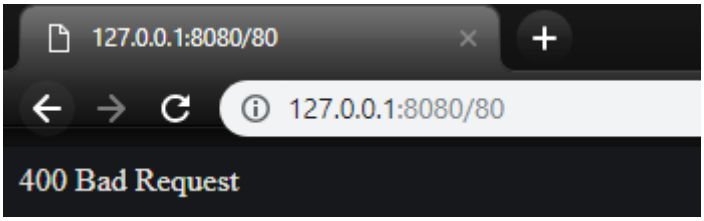
4. 501 Not Implemented

We send **post** request to http server.



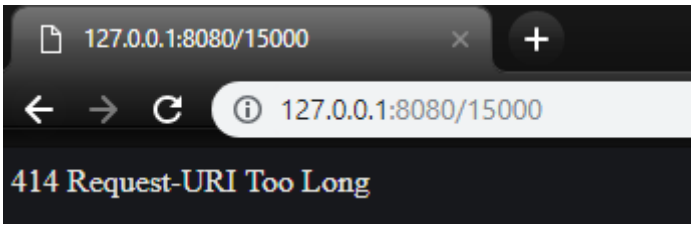
5.400 Bad Request

We try to enter under 100 length.



6. 414 – URI Too Long

We try to enter 15.000 above Proxy size bound



http://ipv4.download.thinkbroadband.com/5MB.zip with Apache Bench Test Results:

A.) 1 request at a time (concurrency level = 1)

```
Server Software:        nginx
Server Hostname:        ipv4.download.thinkbroadband.com
Server Port:            80

Document Path:          /5MB.zip
Document Length:        5242880 bytes

Concurrency Level:      1
Time taken for tests:   65.393 seconds
Complete requests:      10
Failed requests:        0
Total transferred:      52431510 bytes
HTML transferred:       52428800 bytes
Requests per second:    0.15 [#/sec] (mean)
Time per request:       6539.299 [ms] (mean)
Time per request:       6539.299 [ms] (mean, across all co
Transfer rate:          783.00 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       53   90  46.0     77     200
Processing:  4700 6449 2145.3   5604   11507
Waiting:       47   77  40.8     63     178
Total:       4800 6539 2153.5   5673   11644

Percentage of the requests served within a certain time (m
  50%   5673
  66%   5906
  75%   8042
  80%   8194
  90%  11644
  95%  11644
  98%  11644
  99%  11644
 100%  11644 (longest request)
```

B.) 5 requests at a time (concurrency level = 5)

```
Server Software:        nginx
Server Hostname:        ipv4.download.thinkbroadband.com
Server Port:            80

Document Path:          /5MB.zip
Document Length:        5242880 bytes

Concurrency Level:      5
Time taken for tests:   70.041 seconds
Complete requests:      10
Failed requests:        0
Total transferred:      52431510 bytes
HTML transferred:       52428800 bytes
Requests per second:    0.14 [#/sec] (mean)
Time per request:       35020.715 [ms] (mean)
Time per request:       7004.143 [ms] (mean, across all concurrent requests)
Transfer rate:          731.03 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       53  146 107.0    115     379
Processing:  9497 31466 18233.6  31662   65297
Waiting:       47  218 122.9    247     432
Total:       9735 31612 18214.8  32040   65350

Percentage of the requests served within a certain time (ms)
  50%  32040
  66%  35327
  75%  44719
  80%  52780
  90%  65350
  95%  65350
  98%  65350
  99%  65350
 100%  65350 (longest request)
```

C.) 10 requests at a time (concurrency level = 10)

```
Server Software:        nginx
Server Hostname:        ipv4.download.thinkbroadband.com
Server Port:            80

Document Path:          /5MB.zip
Document Length:        5242880 bytes

Concurrency Level:      10
Time taken for tests:   149.234 seconds
Complete requests:      10
Failed requests:        1
   (Connect: 0, Receive: 0, Length: 1, Exceptions: 0)
Total transferred:      52183239 bytes
HTML transferred:       52180529 bytes
Requests per second:    0.07 [#/sec] (mean)
Time per request:       149233.974 [ms] (mean)
Time per request:       14923.397 [ms] (mean, across all concurre
Transfer rate:          341.48 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       47   91  39.2     85     163
Processing: 26980 110864 35976.7 123972  148817
Waiting:       69  420 263.5    416     818
Total:      27027 110955 35989.4 124119  148902

Percentage of the requests served within a certain time (ms)
  50%  124119
  66%  127104
  75%  138838
  80%  140202
  90%  148902
  95%  148902
  98%  148902
  99%  148902
 100%  148902 (longest request)
```

D.) single request at a time (concurrency level = 1) with -k

```
Server Software:        nginx
Server Hostname:        ipv4.download.thinkbroadband.com
Server Port:            80

Document Path:          /5MB.zip
Document Length:        5242880 bytes

Concurrency Level:      1
Time taken for tests:   149.267 seconds
Complete requests:      10
Failed requests:        0
Keep-Alive requests:    10
Total transferred:      52431560 bytes
HTML transferred:       52428800 bytes
Requests per second:    0.07 [#/sec] (mean)
Time per request:       14926.684 [ms] (mean)
Time per request:       14926.684 [ms] (mean, across all concurrer
Transfer rate:          343.03 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0   43 136.1      0     430
Processing:  8023 14884 3915.1  16146   21867
Waiting:       51   86  51.3     68     224
Total:       8023 14927 4001.8  16146   22297

Percentage of the requests served within a certain time (ms)
  50%  16146
  66%  16157
  75%  17110
  80%  17111
  90%  22297
  95%  22297
  98%  22297
  99%  22297
 100%  22297 (longest request)
```

E.) 5 requests at a time (concurrency level = 5) with -k

```
Server Software:        nginx
Server Hostname:        ipv4.download.thinkbroadband.com
Server Port:            80

Document Path:          /5MB.zip
Document Length:        5242880 bytes

Concurrency Level:      5
Time taken for tests:   88.664 seconds
Complete requests:      10
Failed requests:        1
   (Connect: 0, Receive: 0, Length: 1, Exceptions: 0)
Keep-Alive requests:    9
Total transferred:      50684820 bytes
HTML transferred:       50682060 bytes
Requests per second:    0.11 [#/sec] (mean)
Time per request:       44331.828 [ms] (mean)
Time per request:       8866.366 [ms] (mean, across all concurrent requests)
Transfer rate:          558.25 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0   40  45.0     57     113
Processing: 18117 39723 20804.6  39406   81261
Waiting:       64  152  64.0    170     234
Total:      18117 39764 20826.1  39406   81326

Percentage of the requests served within a certain time (ms)
  50%  39406
  66%  39792
  75%  40237
  80%  69783
  90%  81326
  95%  81326
  98%  81326
  99%  81326
 100%  81326 (longest request)
```

F.) 10 requests at a time (concurrency level = 10) with -k

```
Server Software:        nginx
Server Hostname:        ipv4.download.thinkbroadband.com
Server Port:            80

Document Path:          /5MB.zip
Document Length:        5242880 bytes

Concurrency Level:      10
Time taken for tests:   116.556 seconds
Complete requests:      10
Failed requests:        2
   (Connect: 0, Receive: 0, Length: 2, Exceptions: 0)
Keep-Alive requests:    8
Total transferred:      51400672 bytes
HTML transferred:       51397912 bytes
Requests per second:    0.09 [#/sec] (mean)
Time per request:       116555.959 [ms] (mean)
Time per request:       11655.596 [ms] (mean, across all concurrent requests)
Transfer rate:          430.66 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0  108  78.0    120     246
Processing: 15320 87437 38842.6 106160  116016
Waiting:      128  899 500.7    945    1601
Total:      15320 87545 38903.0 106340  116146

Percentage of the requests served within a certain time (ms)
  50%  106340
  66%  107948
  75%  111820
  80%  114122
  90%  116146
  95%  116146
  98%  116146
  99%  116146
 100%  116146 (longest request)
```

When we look at the above test **aspects of concurrency level**:

1.Time taken for tests (seconds)

Concurrency Level: 1

Time taken for tests: 65.393 seconds

Concurrency Level: 5

Time taken for tests: 70.041 seconds

Concurrency Level: 10

Time taken for tests: 149.234 seconds

- This output measures that first socket connection is created to the moment the last response is received. If concurrency level increases, requests will race to reach resources. So, time to return response to all request will increase as seen above.

2.Total transferred (bytes) and HTML transferred (bytes)

Total transferred: 52431510 bytes

HTML transferred: 52428800 bytes

Total transferred: 52431510 bytes

HTML transferred: 52428800 bytes

Total transferred: 52183239 bytes

HTML transferred: 52180529 bytes

- When concurrency level increases, throughput can be occurred because transferred bytes are decreased with 10 concurrency level.

3.Time per request (ms)

Time per request: 6539.299 [ms] (mean, across all concurrent requests)

Time per request: 7004.143 [ms] (mean, across all concurrent requests)

Time per request: 14923.397 [ms] (mean, across all concurrent requests)

- If concurrency level increases, taking time of single request increases as seen above.

4.Requests per second (#/sec)

Requests per second: 0.15 [#/sec] (mean)

Requests per second: 0.14 [#/sec] (mean)

Requests per second: 0.07 [#/sec] (mean)

- When concurrency level increases, server handles requests slowly. Return capacity of server decreases.

5.Transfer rate (Kbytes/sec)

Transfer rate: 783.00 [Kbytes/sec] received

Transfer rate: 731.03 [Kbytes/sec] received

Transfer rate: 341.48 [Kbytes/sec] received

- If concurrency level increases, transfer rate of server decreases because there is resource sharing with multiple access to server.

6.Connection times

Mean: 6539

Mean: 31612

Mean: 110955

- Connection times increases with concurrency level because requests return lately, so packet delivering takes more time anymore.

When we look at the above test **aspects of concurrency level with -k**:

1.Time taken for tests (seconds)

Time taken for tests: 149.267 seconds

Time taken for tests: 88.664 seconds

Time taken for tests: 116.556 seconds

- There is keep alive with increasing concurrency level. So, it increases time of all packet from server according to without keep alive.

2.Total transferred (bytes) and HTML transferred (bytes)

Total transferred: 52431560 bytes

HTML transferred: 52428800 bytes

Total transferred: 50684820 bytes

HTML transferred: 50682060 bytes

Total transferred: 51400672 bytes

HTML transferred: 51397912 bytes

- With increasing concurrency level and keep alive attribute, upload rate of server can be decreased.

3.Time per request (ms)

Time per request: 14926.684 [ms] (mean, across all concurrent requests)

Time per request: 8866.366 [ms] (mean, across all concurrent requests)

Time per request: 11655.596 [ms] (mean, across all concurrent requests)

- With keep alive, server can work slowly while handling the requests.

4.Requests per second (#/sec)

Requests per second: 0.07 [#/sec] (mean)

Requests per second: 0.11 [#/sec] (mean)

Requests per second: 0.09 [#/sec] (mean)

- With keep alive attribute, these values are better than previous.

5.Transfer rate (Kbytes/sec)

Transfer rate: 343.03 [Kbytes/sec] received

Transfer rate: 558.25 [Kbytes/sec] received

Transfer rate: 430.66 [Kbytes/sec] received

- Transfer rates decreases with keep alive.

6.Connection times

Mean: 14927

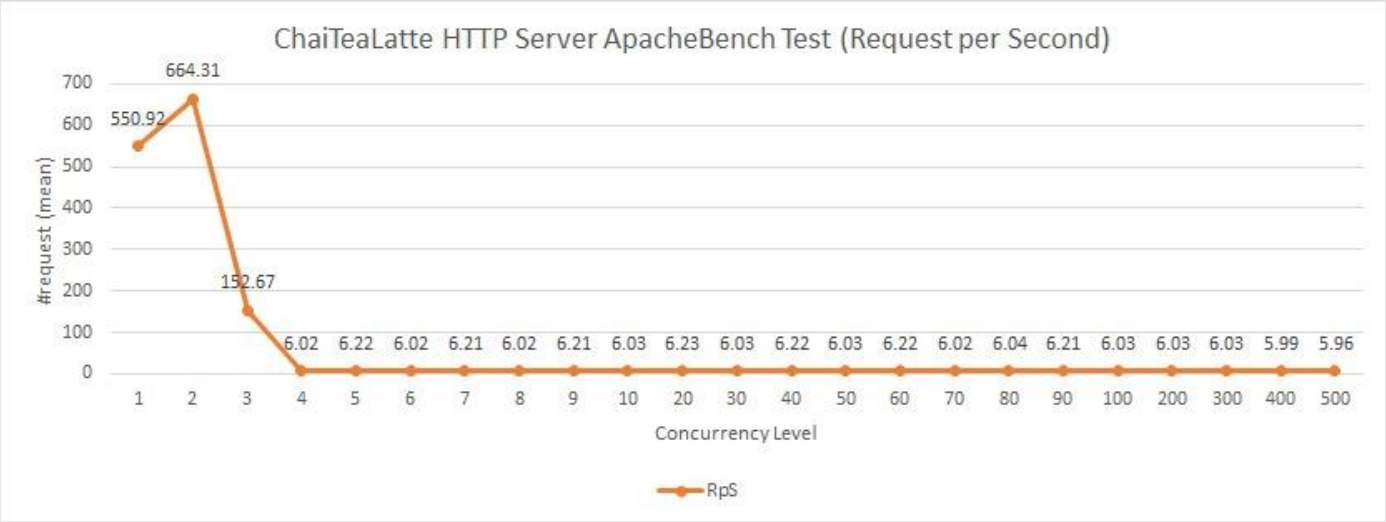
Mean: 39764

Mean: 87545

- Connection times increases with keep alive.

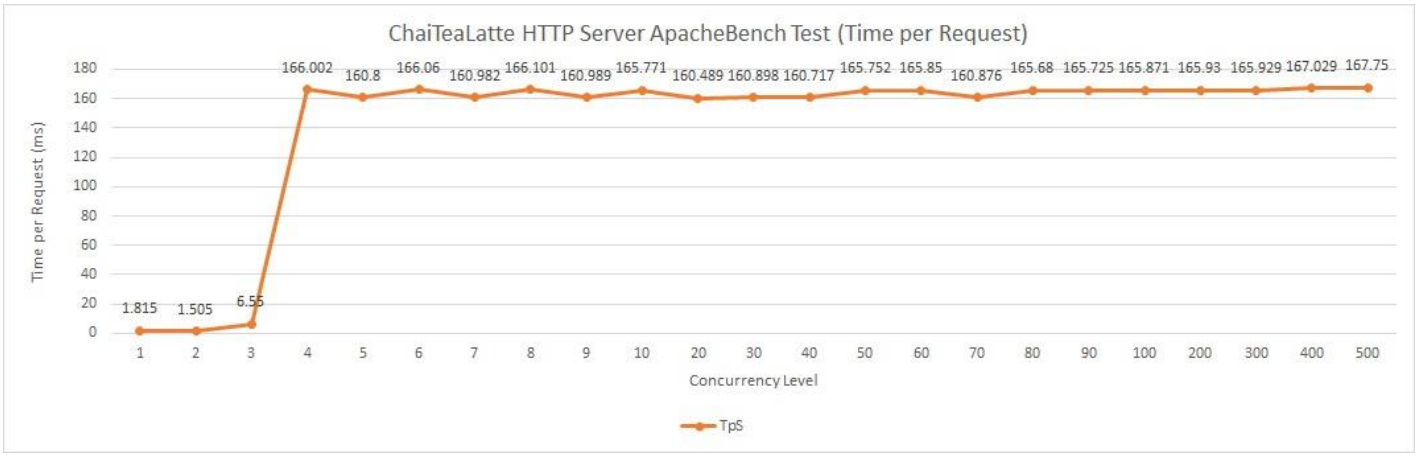
ChaiTeaLatte Server with Apache Bench Test Results:

For the results below, sent 100 requests till concurrency level 100. After that, we sent 500 requests. HTTP server can't handle requests in reasonable time after 500 concurrency level.



Nodes represents 1 second

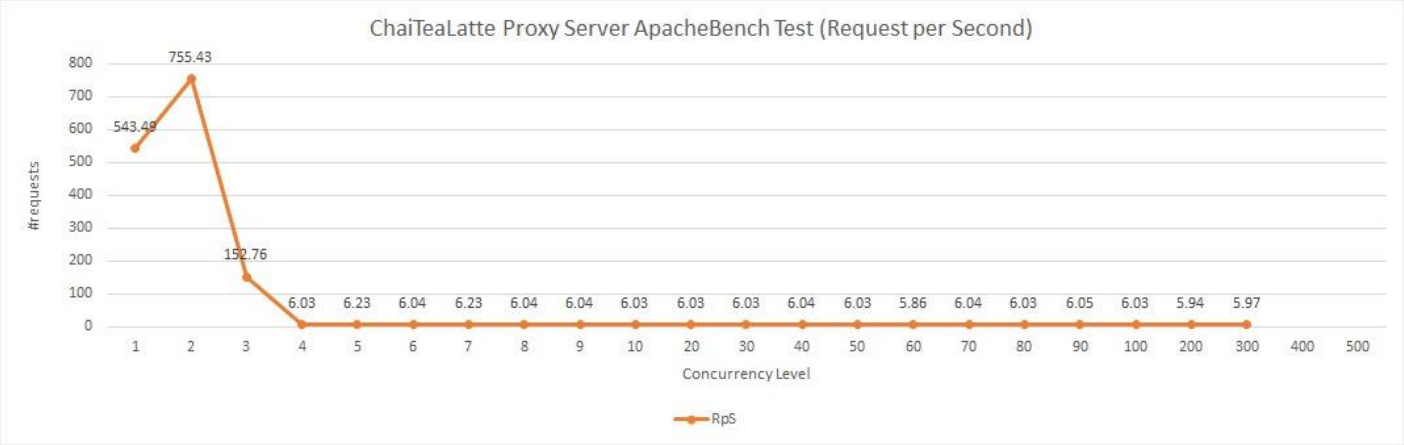
We applied Apache Bench test to our HTTP server. After 1 and 2 concurrency level, there is downfall to 152.67 and approximately 6. Because our HTTP server can't handle requests fast as the very first levels. HTTP server opens CTL recipe file, reads it and turns to client as a response. Besides, our multithread system slows down after level 3 but it has very regular handling rate.



Nodes represents 1 request

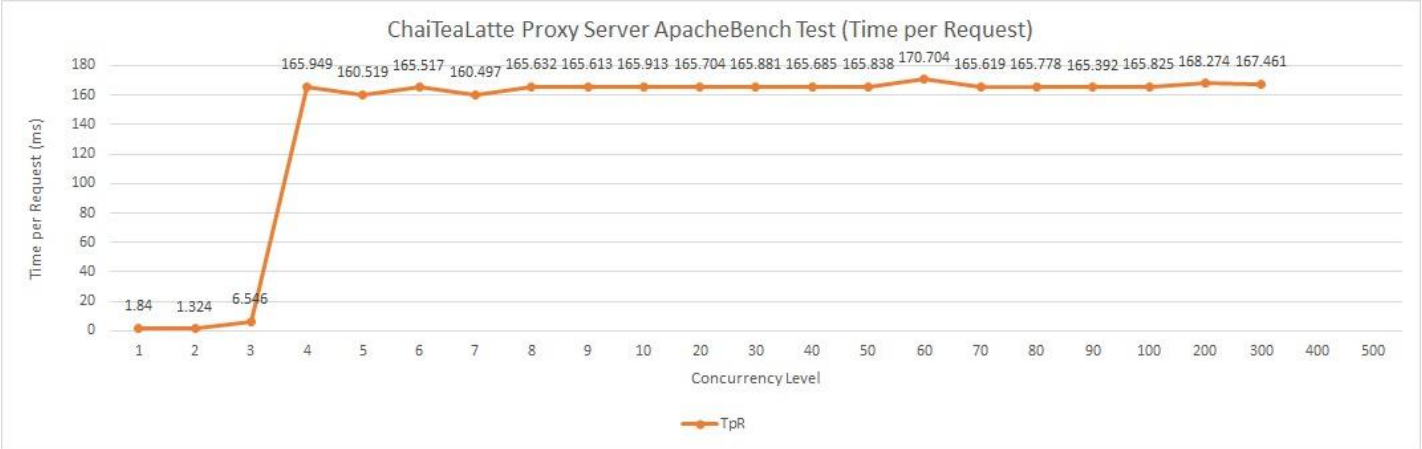
When concurrency level is less than 4, performance of HTTP server is very high. Performance of HTTP server is dropped almost 6 requests after 4 concurrency level.

For the proxy test results, after 300 concurrency level, server can't handle requests in reasonable time.



Nodes represents 1 second

When we test our proxy server, we see that results are like HTTP tests. Because, proxy server holds a cache for requested size of HTML. So, it doesn't need to send a request to HTTP server and wait for a response. Its behavior is similar to HTTP server. Open the cache, read it and send to client. Also multithread system is same.



Nodes represents 1 request

When concurrency level is less than 4, performance of HTTP server is very high. Performance of HTTP server is dropped almost 6 requests after 4 concurrency level.

```
Server Software:        ChaiTeaLatteHTTPServer/1.0
Server Hostname:        127.0.0.1
Server Port:            8080

Document Path:          /3000
Document Length:        3000 bytes

Concurrency Level:      1
Time taken for tests:   0.182 seconds
Complete requests:      100
Failed requests:        0
Total transferred:      313600 bytes
HTML transferred:       300000 bytes
Requests per second:    550.92 [#/sec] (mean)
Time per request:       1.815 [ms] (mean)
Time per request:       1.815 [ms] (mean, across all concurrent requests)
Transfer rate:          1687.20 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.3      0       1
Processing:     1    2   1.0      1       7
Waiting:        0    1   1.0      1       7
Total:          1    2   1.1      1       7

Percentage of the requests served within a certain time (ms)
  50%      1
  66%      2
  75%      2
  80%      2
  90%      3
  95%      4
  98%      6
  99%      7
 100%      7 (longest request)
This is ApacheBench, Version 2.3 <$Revision: 1843412 $>
```

Apache Bench test for HTTP server result with concurrency level 1

```
Server Software:        ChaiTeaLatteHTTPServer/1.0
Server Hostname:        127.0.0.1
Server Port:            8080

Document Path:          /3000
Document Length:        3000 bytes

Concurrency Level:      100
Time taken for tests:   16.587 seconds
Complete requests:      100
Failed requests:        0
Total transferred:      313600 bytes
HTML transferred:       300000 bytes
Requests per second:    6.03 [#/sec] (mean)
Time per request:       16587.097 [ms] (mean)
Time per request:       165.871 [ms] (mean, across all concurrent requests)
Transfer rate:          18.46 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0  166 237.4      0     515
Processing:     3 8304 4861.4   8562   16583
Waiting:        1 8137 4859.0   8061   16582
Total:          4 8470 4858.1   8562   16583

Percentage of the requests served within a certain time (ms)
  50%   8562
  66%  11069
  75%  12574
  80%  13576
  90%  15077
  95%  16081
  98%  16583
  99%  16583
 100%  16583 (longest request)
```

Apache Bench test for Http Server result with concurrency level 100

```
Server Software:        ChaiTeaLatteHTTPServer/1.0
Server Hostname:        127.0.0.1
Server Port:            8888

Document Path:          /3000
Document Length:        3000 bytes

Concurrency Level:      1
Time taken for tests:   0.184 seconds
Complete requests:      100
Failed requests:        0
Total transferred:      313600 bytes
HTML transferred:       300000 bytes
Requests per second:    543.49 [#/sec] (mean)
Time per request:       1.840 [ms] (mean)
Time per request:       1.840 [ms] (mean, across all concurrent requests)
Transfer rate:          1664.43 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.3      0       1
Processing:     0    2   1.5      1      11
Waiting:        0    1   1.4      1      10
Total:          0    2   1.5      1      11

Percentage of the requests served within a certain time (ms)
  50%      1
  66%      2
  75%      2
  80%      2
  90%      3
  95%      5
  98%      5
  99%     11
 100%     11 (longest request)
```

Apache Bench test for Proxy Server result with concurrency level 1

```
Server Software:        ChaiTeaLatteHTTPServer/1.0
Server Hostname:        127.0.0.1
Server Port:            8888

Document Path:          /3000
Document Length:        3000 bytes

Concurrency Level:      100
Time taken for tests:   16.582 seconds
Complete requests:      100
Failed requests:        0
Total transferred:      313600 bytes
HTML transferred:       300000 bytes
Requests per second:    6.03 [#/sec] (mean)
Time per request:       16582.477 [ms] (mean)
Time per request:       165.825 [ms] (mean, across all concurrent requests)
Transfer rate:          18.47 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0  166 237.4      0     514
Processing:     3 8192 4926.3   8022   16580
Waiting:        0 8024 4920.2   8021   16580
Total:          3 8358 4926.9   8523   16580

Percentage of the requests served within a certain time (ms)
  50%   8523
  66%  11029
  75%  12534
  80%  13536
  90%  15066
  95%  16067
  98%  16580
  99%  16580
 100%  16580 (longest request)
This is ApacheBench, Version 2.3 <$Revision: 1843412 $>
```

Apache Bench test for Proxy Server result with concurrency level 100