भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Computer Networks

Assignment -1 - CS3530

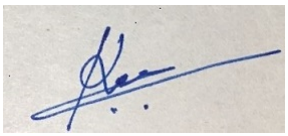
Kartik Srinivas - ES20BTECH11015
October 15, 2022

Plagiarism Statement

Kartik Srinivas

October 15, 2022

We certify that this assignment/report is our own work, based on our personal study and/or research and that we have acknowledged all material and sources used in its preparation, whether they be books, articles, packages, datasets, reports, lecture notes, and any other kind of document, electronic or personal communication. We also certify that this assignment/report has not previously been submitted for assessment/project in any other course lab, except where specific permission has been granted from all course instructors involved, or at any other time in this course, and that we have not copied in part or whole or otherwise plagiarized the work of other students and/or persons. We pledge to uphold the principles of honesty and responsibility at CSE@IITH. In addition, We understand my responsibility to report honor violations by other students if we become aware of it



Kartik Srinivas

Index

1 Implementation Details

Whatever is to be explained is already present inside the *README.md* file There are two ways to run the program If you run *client.py* You will have to enter the requests manually This is an implementation of persistent HTTP . The other way is to use the impersistent HTTP method and just run the shell script *run_tests.sh*

1.1 Internal Implementation

Inorder to send Multiple requests over a connection, there must be something sent to the server that indicates that the connection needs to be closed This END request is sent internally to signify that the connection has ended from the cleint side. This allows graceful termination on the server side. You can see that Wireshark Reecognizes the normal Requests as HTTP requests as well (See the Pcap files). The parsing is done by using the python *string.split()* function.

1.2 Error Handling

There has been usage of Error handling using try-except blocks in python. Appropriate Error codes are also sent back to the client. Here are the Error codes:-

1. HTTP/1.1 200 OK
2. HTTP/1.1 404 Key Not found
3. HTTP/1.1 400 Bad Request

The last one is for the case where the request cannot be parsed. You can test these by running the python file barebones, i.e only run *python client.py* on the xterm window for node h1

2 Basic Topology Analysis

2.1 File Locations

The *Pcap* files for the GET PUT and DELETE requests is under *Real-pcaps/H1-trace-Get-Put-Delete.pcap* and the *Pcap* files for the Get request time analysis are under the folder *GET-request-Analysis*

2.2 GET - Request Time analysis

All times have been calculated using *Wireshark* Timestamps on the client side. (**quick trick**: use keystrokes control + T to get time differences quickly) The GET Requests use Impersistent HTTP Method

Key	Request 1	Request 2	Request 3
key1	0.0009	0.0011	0.00087
key2	0.0014	0.0008	0.0006
key3	0.0014	0.001	0.0007
key4	0.0009	0.001	0.0006
key5	0.001	0.002	0.0007
key6	0.0009	0.0009	0.0012
Average	0.0010833	0.001333	0.00076

3 Star Topology Analysis

3.1 File Locations

The *Pcap* files for the GET request for key-1 for all three interfaces *s1-eth1, s1-eth2, s1-eth3* is under *Real-pcaps/H1.pcap, H2.pcap, H3.pcap* and the *Pcap* files for the Get request time analysis are under the folder *GET-request-Analysis*

3.2 GET - Request Time analysis

The GET Requests use Impersistent HTTP Method

3.3 GET Request Time analysis

This is done through Impersistent HTTP Requests.
The files that contain the pcap traces for the numbers are *First-Time.pcap, Second-Time.pcap, Third-Time.pcap*

Key	Request 1	Request 2	Request 3
key1	0.00185	0.002	0.0081
key2	0.004	0.006	0.00075
key3	0.0042	0.001	0.0005
key4	0.004	0.001	0.0008
key5	0.005	0.004	0.0009
key6	0.005232	0.0008	0.0015
Average Time	0.004047	0.0001566	0.00087667

3.4 Reasons for Results

As you can see the average time goes down after running the same GET requests for the second and third time. This is because the cache server **Stores the key's that are requested recently**. In the beginning only key1 is inside the cache. Hence **initally the time for key1 is low**, and that of the other keys is high. But once all the requests have been made for the first time, and we move on to the second time, the key's are already in the cache!. Hence the time taken for the cache to return the key is lower since it does not have to query the server for the keys that it does not have.

$$t_{client-cache-client} < t_{client-cache-server-cache-client}$$

Requests 2 and 3 are the ones where only the Cache has been queried , and hence the time taken by them is lesser.