COL374/672 Computer Networks: 2020-21 semester I

Assignment 3

In this assignment, we will experiment with different strategies to quickly download a large file.

A large text file of about 6.5MB is available at http://vayu.iitd.ac.in/big.txt and http://norvig.com/big.txt. Some details are given below:

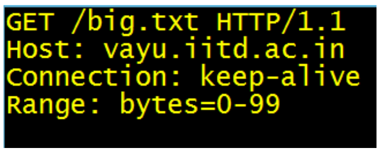
MD5 sum: 70a4b9f4707d258f559f91615297a3ec

Size: 6488666 bytes

We want to develop a tool to download this file quickly, and also be resilient to disconnections, Eg. if your network is poor and disconnects while a file transfer is in progress then we want that the tool should resume downloading the file from the point of last disconnection and not download the same portions twice.

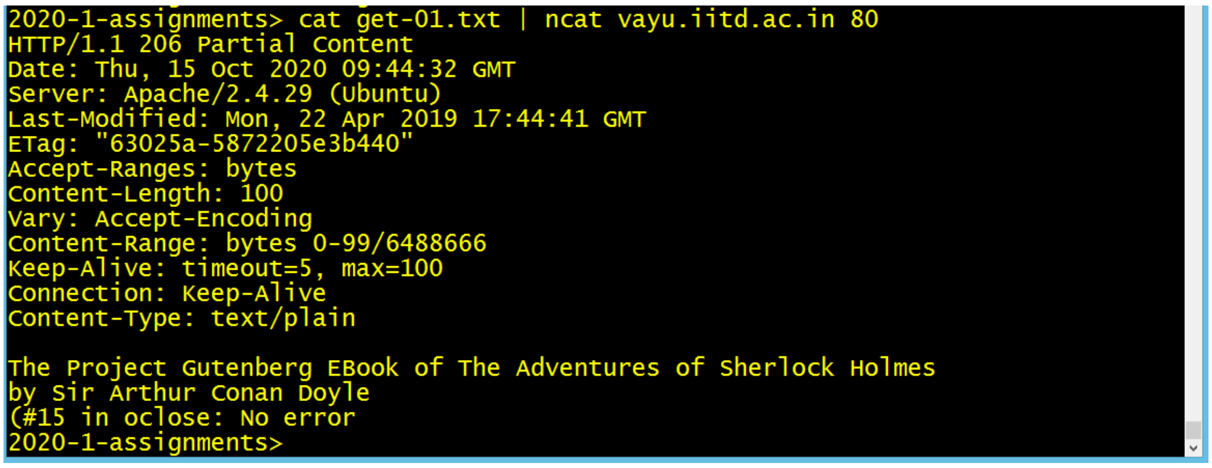
1. Start with writing a simple program using TCP sockets that opens a TCP connection to the server, issues a GET request, downloads the entire content, and stores it in a file, in one go. Make sure you are able to download the file correctly and the MD5 sum matches. Note that you will have to read byte by byte.

2. Now try downloading the file in parts, by sending an HTTP GET command to download a particular range of bytes, using the Range header in HTTP. The HTTP request will look as follows:



```
GET /big.txt HTTP/1.1
Host: vayu.iitd.ac.in
Connection: keep-alive
Range: bytes=0-99
```

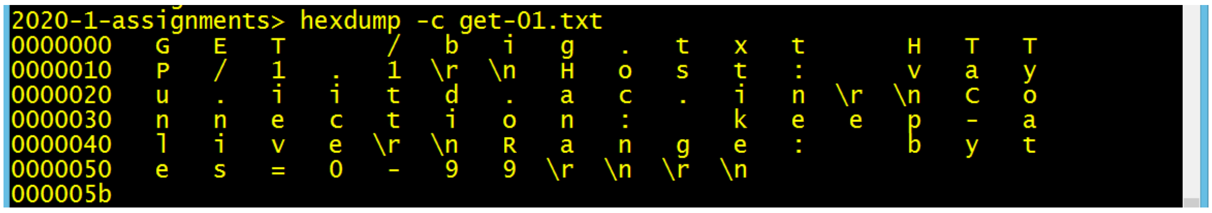
You will find it easy to experiment by creating a small file containing this request, using a text editor like vim. Use it in the binary mode with "vim –b". Look up vim commands to learn how to edit the file to insert special characters. You can then use the ncat command as follows:



```
2020-1-assignments> cat get-01.txt | ncat vayu.iitd.ac.in 80
HTTP/1.1 206 Partial Content
Date: Thu, 15 Oct 2020 09:44:32 GMT
Server: Apache/2.4.29 (Ubuntu)
Last-Modified: Mon, 22 Apr 2019 17:44:41 GMT
ETag: "63025a-5872205e3b440"
Accept-Ranges: bytes
Content-Length: 100
Vary: Accept-Encoding
Content-Range: bytes 0-99/6488666
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/plain

The Project Gutenberg EBook of The Adventures of Sherlock Holmes
by Sir Arthur Conan Doyle
(#15 in oclose: No error
2020-1-assignments>
```

A hexdump of this file will look like the following. Notice the \r and \n special characters. In vim, you will need to insert \r manually, the \n will be inserted by vim automatically for each newline.



```
2020-1-assignments> hexdump -c get-01.txt
0000000   G   E   T       /   b   i   g   .   t   x   t       H   T   T
0000010   P   /   1   .   1  \r  \n   H   o   s   t   :       v   a   y
0000020   u   .   i   i   t   d   .   a   c   .   i   n  \r  \n   C   o
0000030   n   n   e   c   t   i   o   n   :       k   e   e   p   -   a
0000040   l   i   v   e  \r  \n   R   a   n   g   e   :       b   y   t
0000050   e   s   =   0   -   9   9  \r  \n  \r  \n
000005b
```

For further debugging, you can also run a Wireshark in parallel to check the exact bytes being sent across the network. If you send an incorrectly formatted request then the server is likely to reply with an error.

3. After you are able to send requests to download the file in parts, you can begin experimenting in all sorts of interesting ways. First decide on a chunk-size, say 10KB, to download the file in chunks of that size. Your program can keep track of which chunks have been downloaded or not. Use threads to now experiment with:

   - The number of parallel TCP connections to open to the server.

   - Needless to say that you should not open a new TCP connection for each chunk. You can issue multiple GET requests on the same TCP connection, one after the other, and the server will send back the corresponding chunks in the same order as the GET requests.

     A straightforward manner to set this up would be to have multiple threads, with one TCP connection per thread, and each connection used to download a large number of chunks, with no chunk downloaded in duplicate on another connection.

     Your program will have to do some basic bookkeeping for this purpose, to make sure that a chunk is assigned to only one connection for download. Of course, one way is to pre-assign chunks to connections (or threads). Another way is that whenever a thread completes a chunk download, it queries a synchronized object to pick up a new chunk. Chunks will thus get allocated to threads in a dynamic manner.

   - Does your download time keep decreasing with more and more parallel TCP connections? Try to explain your finding. You can also log the download progress for each connection, and draw a graph that shows on the y-axis the number of bytes downloaded, and on the x-axis shows the time. You can check if some connections are faster than others, or some connections get stalled for some reason.

   - You can even spread your connections between vayu.iitd.ac.in and norvig.com by downloading some chunks from one server and some from the other. Does your download time reduce further if you download from different sources in parallel? What does this tell you about where bottlenecks lie? Is one server faster than the other? Is your program able to use this to download more from the faster server?

Note that eventually you should also be capable of reassembling the chunks to reconstruct the entire file, and that the MD5 sum should match the original.

4. Now you can get bolder. Restructure your program so that in case a TCP connection breaks, the thread does not end but tries to open a new connection. Whenever a new connection succeeds, the thread resumes to download more chunks.

   You can emulate connection failures by disconnecting your network, after a while the TCP connection will timeout and raise an exception. You can catch the Exception and try to initiate a new connection. The default TCP connection timeout may be very large, therefore you should set a short enough timeout to detect such connection failures.

What to submit:

- A neat report in pdf, formatted using Latex. Please use the same question numbering as above.
- The final program at the last stage, which is resilient to disconnections, and takes as input a CSV file with the following structure:

  [URL-1 for the object], [Number of parallel TCP connections to this URL]

  [URL-2 for the object], [Number of parallel TCP connections to this URL]

  …

Congratulations! This is effectively what BitTorrent does – it obtains a torrent file that contains a list of peers hosting the file, and downloads different chunks of the file from different peers.