CS/ECE 438: Computer Networks **Spring 2026**

Machine Problem 1

Handed Out: February 2nd, 2026 *Due: 11:59pm, February* 15th, 2026

TA: Martin Chong

Abstract

This machine problem introduces you to a bare-bones HTTP client that can get data from any web server. This is the kind of code that is running in your browser. You will also create a HTTP server that can serve data to other clients much like how a real server would function.

1 Introduction

In this assignment, you will implement a simple HTTP client and server.

For an overview on HTTP, please refer to the following document:

`https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/Overview`

The client should be able to correctly issue `GET` requests to standard web servers, and standard web browsers should be able to correctly issue `GET` requests to your server.

The test setup consists of two virtual machines: one running the client and one running the server. Each test will use either your client or `wget`, and either your server or `thttpd`. Your client does *not* need to support caching or recursive retrieval of embedded objects.

HTTP runs on top of TCP. You may use Beej's `client.c` and `server.c` examples as a starting point.

Your server must support concurrent connections: if one client is downloading a 10MB object, another client that comes looking for a 10KB object shouldn't have to wait for the first to finish.

2 What is expected in this MP?

2.1 HTTP Client

Your client should run as

```
./http_client http://hostname[:port]/path/to/file
```

e.g.

```
./http_client http://127.0.0.1/index.html
./http_client http://illinois.edu/index.html
./http_client http://12.34.56.78:8888/somefile.txt
./http_client http://localhost:5678/somedir/anotherfile.html
```

Please note that the URL passed to the http client might be arbitrarily long and might have arbitrarily many levels in its path, as long as it's in well-formed HTTP URL format.

If there is no :port, assume port 80 – the standard HTTP port. **You should write the file that you receive to a file called "output" (no file extension, like txt or html).** Here's the very simple HTTP GET that wget uses:

```
GET /test.txt HTTP/1.1
User-Agent:  Wget/1.12 (linux-gnu)
Host:  localhost:3490
Connection:  Keep-Alive
```

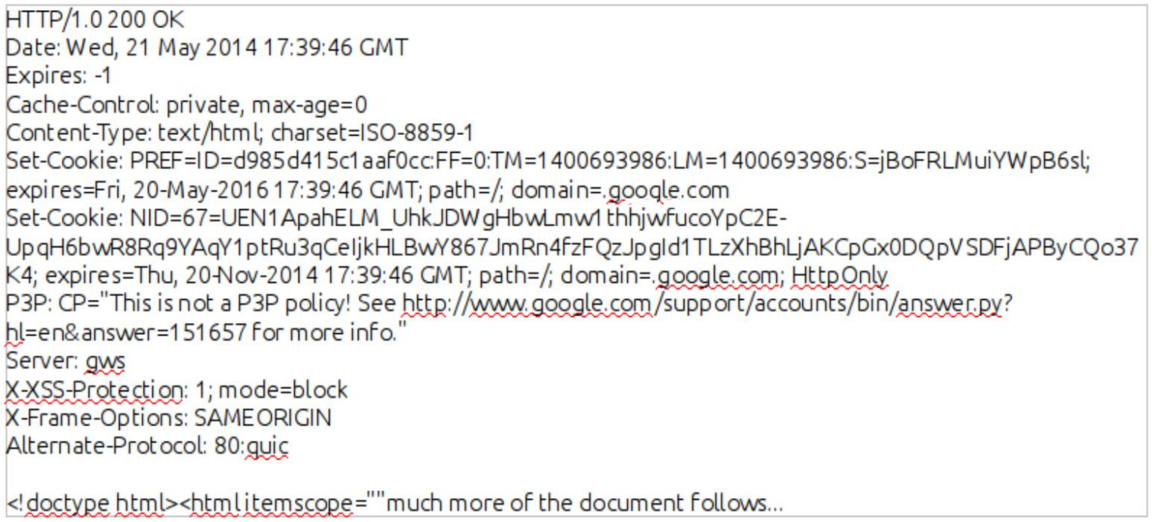
The GET /test.txt instructs the server to return the file called test.txt in the server's top-level web directory. User-Agent identifies the type of client. Host is the URL that the client was originally told to get from – exactly what the user typed. This is useful in case a single server has multiple domain names resolving to it (maybe cs.illinois.edu and math.illinois.edu), and each domain name actually refers to different content. This could be a bare IP address, if that's what the user had typed. The 3490 is the port – this server was listening on 3490, so I called "wget localhost:3490/test.txt". Finally, Connection: Keep-Alive refers to TCP connection reuse, which will be discussed in class.

Note that the newlines are technically supposed to be CRLF – so, "\r\n" on a Unix machine.

Only the first line is essential for a server to know what file to give back, so **your HTTP GETs can be just that first line**. HTTP specifies that the end of a request should be marked by a blank line — so **be sure to have two newlines at the end**. (This is necessary because TCP presents you with a stream of **bytes**, rather than **packets**.)

2.2 HTTP Server

Now for the HTTP response. Here's what Google returns for a simple GET of /index.html:



```
HTTP/1.0 200 OK
Date: Wed, 21 May 2014 17:39:46 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
Set-Cookie: PREF=ID=d985d415c1aaf0cc:FF=0:TM=1400693986:LM=1400693986:S=jBoFRLMuiYWpB6sl;
expires=Fri, 20-May-2016 17:39:46 GMT; path=/; domain=.google.com
Set-Cookie: NID=67=UEN1ApahELM_UhkJDWgHbwLmw1thhjwfucoYpC2E-
UpqH6bwR8Rq9YAqY1ptRu3qCeIjkHLBwY867JmRn4fzFQzJpgId1TLzXhBhLjAKCpGx0DQpVSDFjAPByCQo37
K4; expires=Thu, 20-Nov-2014 17:39:46 GMT; path=/; domain=.google.com; HttpOnly
P3P: CP="This is not a P3P policy! See http://www.google.com/support/accounts/bin/answer.py?
hl=en&answer=151657 for more info."
Server: gws
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN
Alternate-Protocol: 80:quic

<!doctype html><html itemscope=""much more of the document follows...
```

Your server's headers will be much simpler (but still correct and complete): only include the response code. When correctly returning the requested document, use `HTTP/1.1 200 OK`, like this example. When the client requests a non-existent file, return `HTTP/1.1 404 Not Found`. Note that you can still have document text on a 404 – allowing for nicely formatted or more informative "whoops, file not found!" messages. For any other errors, you may simply return `400 Bad Request`.

An important note: see how there's a blank line between the header and document text in the Google response? That's a well-defined part of the protocol, marking the end of the header. Your server must include this blank line. Again, HTTP newlines are CRLF. Your server should take the port to run on as a command line argument, and should treat all filepaths it's asked for as being relative to its current working directory. (Meaning just pass the client's request directly to `fopen`: if the client asks for `GET /somedir/somefile.txt`, the correct argument to `fopen` is `somedir/somefile.txt`). Your server executable should be called http_server, e.g.:

```
sudo ./http_server 80
./http_server 8888
```

(The sudo is there because using any port <1024 requires root access.)

3 Autograder and Submission

Similar to the MP0, checkout your mp1 directory from the class repository. **Do not copy & paste files yourself. Make sure to merge release as shown below. Otherwise, the autograder may not recognize your repository.**

```
git pull

git fetch release

git merge release/main -m ‘‘Merging release repository’’
```

The contents of the checked out mp1 folder will be very similar like mp0 when you first checked it out. Use those programs as a starting point and make the modifications required for this assignment.

Modify the Makefile such that a simple `make` command in your mp1 folder creates the required executables. The autograder will simply call `make` to compile your code. Be careful about the executable filenames and output filenames.

3.1 config.ini

You have also received the config file, `config.ini`, which is unique for every MP. It should have the following format:

```
[MP]
version = <version number>
```

the `version` number applies as in MP0. Increment it to a value greater than 0 so the autograder can grade your code. Increment the version number every time you make changes and want a re-grading. From MP1 onwards, the latest version submission will be used for your final grade. Do not edit your MP code after you have made your final submission.

3.2 logs.txt and score_history.txt

These two files are in the **_grades** branch and can be used to check your previously submitted versions, scores, and the number of extra days used.

3.3 Code submission

Follow the git instructions from MP0 to submit your code. Once the autograder is enabled, you will be able to run the `./see_results.sh` script to get the results.

Tests generally take 1-4 minutes, and there may be a queue of students.

Caution: During the hours leading up to the submission deadline the queues could be multiple hours long. So it is advisable to get your work done early.

PLEASE do not fall into the trap of “debugging on the autograder”. If you submit a new version every time you make some change that might help pass an extra test, you are going to waste a lot of time waiting for results. Rather, only submit when you have made major progress or have definitively figured out what you were previously doing wrong. If you aren’t genuinely surprised that your most recent submission didn’t increase your score, you are submitting too often.

4 Grade Breakdown

- 25%: you submitted your assignment correctly and it compiles correctly on the autograder (You must have at least successfully committed your files into git to benefit from this.)
- 25%: wget can retrieve files from your HTTP server
- 25%: your client can retrieve files from your HTTP server
- 25%: your server does concurrency correctly: 1 very long download does not block many smaller downloads from starting immediately.

We will use `diff` to compare the server’s copy with the downloaded copy, and you should do the same. If `diff` produces any output, you aren’t transferring the file correctly.

5 Notes

You must use C or C++.
Common mistake: libraries must go at the end of the compile command.

Your program must have a Makefile; running "make" should build all executables.

Do not make your github repo public to avoid plagiarism issues. You will be held partially responsible for any resultant plagiarism.

Your code must be your own. You can discuss very general concepts with others, but if you find yourself looking at a screenful/whiteboard of pseudocode (let alone real code), you’re going too far.

Refer to the class slides and official student handbook for academic integrity policy. In summary, the standard for guilt is "more probable than not probable". Any violations of academic integrity will be reported and incur a minimum penalty of one letter grade AND a zero on the MP.

You can use libraries from wherever for data structures. You MUST acknowledge the source in a README. Algorithms (e.g. Dijkstra's) should be your own.

Your code must run on the autograder setup, which runs on Ubuntu 24.04 LTS.

Input files on the grader are READ-ONLY. Do not use the "rb+" mode to read them; the "+" asks for write permission. (In general, you shouldn't use "rb+" unless you need it, which should be rare.)

Input files on the grader are general binary data, NOT text.

If you run the see_results.sh file on mp1 before the autograder has been activated, your directory will move to the _grades branch. You will have to manually execute `git checkout master` to get back to your working branch. DO NOT work on the _grades branch!

All of your source files will be checked for plagiarism. So do not use pieces of code outside of what has been provided in `_release`.