CSc 361: Computer Communications and Networks (Spring 2018)

Programming Assignment 1: Smart Web Client

Spec Out: Jan 5, 2018
Final Due: 23:55 pm, Jan 26, 2018

1 Goal

The project is to build a tool at web client to collect information regarding a web server. The purpose of this project is two fold:

- to provide students with hands-on experience with socket programming **in Python**,
- to help students understand the application-layer protocols HTTP/HTTPs. Note that HTTPs is not a standalone protocol, but instead it is HTTP over Transport Layer Security (TLS). In this assignment, your main focus is on HTTP, not TLS.

2 Background

2.1 HTTP

HTTP stands for Hyper Text Transfer Protocol and is used for communication among web servers.

The web client initiates a conversation by opening a connection to a web server. Once a connection is set up, the client sends up an HTTP request. The server sends an HTTP response back to the client and closes the connection. An HTTP request consists of two parts: a header and a body. Whether a body follows a header or not is specified in the header.

Using *single-line header of HTTP request* as an example, the first line of any request header should be:

- the method field: The method field can take on several different values, including GET, POST, HEAD, and so on.
- the URL field: It is the field to identify a network resource, e.g., "http://www.csc.uvic.ca/index.html".
- the HTTP version field: This field is "HTTP/1.0".

An example is: "*GET http://www.csc.uvic.ca/index.php HTTP/1.0*". The request header and body are separated by two sets of a carriage return and a linefeed. Since we do not need the body, the end of a header marks the end of a request. Using a C char string as an example, the request above should be: "*GET http://www.csc.uvic.ca/index.php HTTP/1.0\r\n\r\n*". Please be careful when you use Python String.

The response from a server also has two parts: a header and a body. The first line of a header should be:

- the HTTP version field,
- the status code field,
- the phrase field.

Two main status codes include 200 and 404. The status code 200 means that the request succeeded and the information is returned in the response. The status code 404 means that the requested document does not exist on this server. Two example response messages are: "*HTTP/1.0 404 Not Found*\r\n\r\n" and "*HTTP/1.0 200 OK*\r\n\r\n *data data data ...*" Another two status codes 505: "HTTP Version Not Supported", and 302: "302 found" for URL redirection are also useful for this assignment.

2.2 URI

URI stands for Uniform Resource Identifier and is also known as the combination of Uniform Resource Locators (URL) and Uniform Resource Names (URN). It is a formatted string which identifies a network resource. It generally has the format: *protocol://host[:port]/filepath*. When a port is not specified, the default HTTP port number is 80, and the default HTTPS port number is 443.

2.3 Cookies

An HTTP cookie is a small piece of data that a server sends to the user's web browser. The browser may store it and send it back with the next request to the same server. Typically, it's used to tell if two requests came from the same browser keeping a user logged-in, for example. It remembers stateful information for the stateless HTTP protocol. Cookies have many applications in web, such as tracking, authentication, and web analytics. Due to this reason, cookies also cause many concerns on security and privacy breach.

The textbook includes simple introduction on cookies. More detailed information could be found at: *https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies*. Python includes dedicated modules to handle Cookies: *https://docs.python.org/3/library/http.cookies.html*. Nevertheless, you are no allowed to use this package because it defeats the purpose of a network-course assignment.

3 Project Description

You are required to build a smart web client tool, called *SmartClient*, in Python. **Note that for consistence, program in other language will not be accepted!**

Given the URL of a web server, your *SmartClient* needs to find out the following information regarding the web server:

- 1. whether or not the web server supports HTTPs,
- 2. the newest HTTP version that the web server supports (http1.0, http1.1, or http2.0),

- 3. the name, the key, and the domain name of cookies that the web server will use.

Your program first accepts URI from stdin and parses it. Then it connects to a server, sends an HTTP request, and receives an HTTP response. You should also implement a routine that prints out the response from the server, marking the header and the body. When you finish the client, you can try to connect to any HTTP server. For instance, type "https://www.uvic.ca/" as the input to the client program and see what response you get.

As an example output, after you run your code with

```
% SmartClient http://www.uvic.ca
```

Your *SmartClient* may output the received response from the server (**optional**), e.g.,

```
---Request begin---
GET http://www.uvic.ca/index.html HTTP/1.1
Host: www.uvic.ca
Connection: Keep-Alive

---Request end---
HTTP request sent, awaiting response...

---Response header ---
HTTP/1.1 200 OK
Date: Tue, 02 Jan 2018 22:42:27 GMT
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Set-Cookie: SESSID_UV_128004=VD3vOJhqL3YUbmaZSTJre1; path=/; domain=www.uvic.ca
Set-Cookie: uvic_bar=deleted; expires=Thu, 01-Jan-1970 00:00:01 GMT; Max-Age=0; path=/; dom
Keep-Alive: timeout=5, max=100
Connection: close
Content-Type: text/html; charset=UTF-8
Set-Cookie: www_def=2548525198.20480.0000; path=/
Set-Cookie: TS01a564a5=0183e07534a2511a2dcd274bee873845d67a2c07b7074587c948f80a42c427b1f7ea
Set-Cookie: TS01c8da3c=0183e075346a73ab4544c7b9ba9d7fa022c07af441fc6214c4960d6a9d0db2896a8c
Set-Cookie: TS014bf86f=0183e075347c174a4754aeb42d669781e0fafb1f43d3eb2783b1354159a9ad8d81f7

--- Response body ---
Body Body .... (the actual content)

```

Note that some lines in above output were truncated.

Your code might need to send multiple requests in order to find out the required information. Your code should output the final results (**mandatory**), for example:

```
website: www.uvic.ca
1. Support of HTTPS: yes
2. The newest HTTP versions that the web server supports: HTTP/1.1
```

```
3. List of Cookies:
name: -, key: SESSID_UV_128004,  domain name: www.uvic.ca
name: -, key: www_def, domain name: www.uvic.ca
name: -, key: TS01a564a5, domain name: www.uvic.ca
name: -, key: TS01c8da3c, domain name: www.uvic.ca
name: -, key: TS014bf86f, domain name: uvic.ca
name: Google Analytics, key: __utma, domain name: uvic.ca
name: Google Analytics, key: __utmb, domain name: uvic.ca
name: Google Analytics, key: __utmc, domain name: uvic.ca
name: Google Analytics, key: __utmz, domain name: uvic.ca
name: TradeDesk, key: TDID, domain name: .adsrvr.org
```

3.1 Other Notes

1. Regarding other printout: Anything not specified in Assignment 1 is optional. For example, you can decide whether or not to print out the IP address, port number, and so on. When TAs test your code, if your code works fine without any problem, you are fine even if you do not print out anything not required in Assignment 1. Nevertheless, if your code does not work, TAs will not spend time to figure out what is wrong and you get a zero mark on the required function (Refer to the table in Section 5 of Assignment 1). In this case, if your code includes some printout to show intermediate results, TAs will have an idea on how far you have achieved and give you some partial mark based on their own judgement.

2. Regarding readme file. Readme file is important. Without it TAs will not know how to compile your code and how to run your code. It would waste our time to deal with your complaint if TAs cannot run your code and give you a zero.

3. For more information on HTTP, HTML, URI, etc., please refer to http://www.w3.org. It is the home page of W3 Consortium and you will find many useful links to subjects related to the World Wide Web.

4 Schedule

In order to help you finish this programming assignment successfully, the schedule of this assignment has been synchronized with both the lectures and the lab tutorials. Before the final deadline, there are three lab sessions arranged during the course of this assignment. A schedule is listed as follows:

Session	Tutorial	Milestones
Lab 1	lab environment, P1 spec go-through, design hints, python	design and code skeleton
Lab 2	socket programming and testing, DNS	alpha code done
Lab 3	socket programming and last-minute help	beta code done and demo

5 Deliveries and Marking Scheme

For your final submission of each assignment you are required to submit your source code to connex. You should include a readme file to tell TA how to compile and run your code. At the last lab session that you attend, you need to demo your assignment to TAs. Nevertheless, before the final due date, you can still make changes on your code and submit a *change.txt* file to connex to describe the changes after your demo.

The marking scheme is as follows:

Components	Weight
Error handling	10
Correct output for "support of HTTPS"	10
Correct output for "the newest HTTP version that the web server supports"	30
List of Cookies	40
Code style	5
Readme.txt and change.txt(if any)	5
Total Weight	100

Important Note: listing cookies is a very tricky business, and it is possible that you will not get a unique, static answer due to the dynamic changes in cookies, some created dynamically based on users' interactive input. This assignment requirement, while not being ideal for a third-year course, is intended to push your limit for doing research yourself. As a benchmark, we will use some online tool, http://www.cookie-checker.com/, to validate your result. You will get full mark on the "list of cookies (40%)", as long as your results have over 75% similarity with the benchmark results.

6 Plagiarism

This assignment is to be done individually. You are encouraged to discuss the design of your solution with your classmates, but each person must implement their own assignment.

The End