

# Mini-Project

---

**Due** Dec 5, 2022 by 11:59pm    **Points** 18    **Submitting** a file upload    **File Types** zip  
**Available** Oct 26, 2022 at 11:59pm - Dec 9, 2022 at 11:59pm

---

This assignment was locked Dec 9, 2022 at 11:59pm.

In this Mini Project you will practice applying your **Socket Programming** knowledge. You have two options to either build a simple web/web proxy server or implement your high performing transport layer protocol. You will practice:

- Socket Programming
- Connection Management
- Protocol Implementation & Analysis

**Estimated Required Time:** Total of 6-15 hours per person\*

\*Based on level of previous practice with python socket programming and the involved protocols, and the choice of the project.

You have the option to choose between two project choices. Option one which is implementation of a web server is the easier, but more well-defined project. The first options includes a bonus optional deliverable, and the second option, which is your choice of reliable transport protocol needs further planning and specification on your side, and therefore has 2 bonus points incorporated to its total grade. The total MP grade of option one is out of 20 [+2 bonus points] and option two is out of 22, which means possibility of 10% bonus points on both items, but with different ways of earning them.

Reminder: Step one was your team selection, which you already have submitted. So, what we will explain here will start from Step Two for both options.

## Option One: Web & Web Proxy Servers


### Step Two (10 points):

(a) You have seen socket programming at the end of Module (2) and will code simple TCP and UDP client and servers in IS(3) and IS(4) using socket programming. In the first part of your mini project, using what you learned about socket programming, and HTTP protocol, create your simple web server. (7 points)

At this step, your web server will handle one HTTP request at a time, and implements the following messages for all of the relevant methods to the client:

Code	Message
------	---------

200	OK
304	Not Modified
400	Bad request
404	Not Found
408	Request Timed Out

(b) To test your web server, copy [test.html](https://canvas.sfu.ca/courses/70733/files/19046551/download?wrap=1) (<https://canvas.sfu.ca/courses/70733/files/19046551/download?wrap=1>)  ([https://canvas.sfu.ca/courses/70733/files/19046551/download?download\\_frd=1](https://canvas.sfu.ca/courses/70733/files/19046551/download?download_frd=1)) in the same directory of your web server. Then find out the IP address of your machine, and port used in the code for web server and type the following in your web browser:

```
http://IP_ADDRESS:PORT/test.html
```

Edit your test file, or send the request properly to test your implementation for different message scenarios. Print screen, or cut and past output and document your test procedures. (3 pts)

### Step Three (8 point):

(a) For the second part of your mini project, think about a web proxy server. What is different in request handling in a proxy server and a web server that hosts your files? Write down the detailed specifications you come up with for a minimal proxy server only using the knowledge you have from module (2) slides 29-34 (2 points), and implement them (4 points).

(b) Decide the test procedure to show the working of your proxy server. Does this need possible changes at the client side? If your answer is yes, you are not required to implement them, but need to describe and find alternative ways to test your server side functionality.

Print screen, or cut and past output and document your test procedures to show. (2 points)

### Bonus: Step Four (2 points):

(a) For the bonus part of your mini project, extend your web server (from step two) to its multi-threaded version, capable of handling multiple requests simultaneously. You can implement this with your main thread listening for clients at a fixed port. When your server receives a TCP connection request from a client, the TCP connection to serve the client should happen through another port. Each TCP connection requested is handled in a separate thread. (1 point)

(b) Decide the test procedure to show the multi-threaded nature of your server. Print screen, or cut and past output and document your test procedures to show. (1 point)

### Deliverables

For steps Two to five, please make a zip file including:

- Your single thread web server code (.py)
- Your web proxy server code (.py)
- Your multi-threaded web server code (.py)
- Your (if any) modified test files (.html)
- A .pdf file including specifications of proxy requirements and your test procedures for all steps, and print screens

and submit it to this activity in Canvas. One submission will be enough for each group (should automatically show for both group members).

## Option Two: Your Transport Protocol

### Step Two (12 points):

TCP is the reliable transport protocol of the Internet. You will study and observe the TCP protocol behaviour in IS(4). In this step of your project, we want you to implement your own flavor of connection-oriented, reliable, pipelined transport layer protocol which includes flow control and congestion control mechanisms as well. To do this, you need to specify your mechanisms, and decide your socket type (2 points). On the chosen socket, you then define and implement your own transport layer protocol. (10 points)

### Step Three (3 points):

In this step of the project, you need to analyze the traffic of your transport protocol using wireshark. Through this traffic analysis, you need to prove that your protocol is connection-oriented, has acceptable performance, and implements flow control and congestion control mechanisms.

### Step Four (5 points):

Test your protocol with other traffic (4 points). You can choose to use network simulators, run your protocol across the network, or bring up your virtual nodes to test. Is your protocol fair? Can you prove? (1 point)

## Deliverables

For steps Two to four, please make a zip file including:

- Your code (.py)
- Your capture and analysis (e.g., figure) files
- A .pdf file including specification of your protocol, your test procedures, and explanation of your results showing the desired working of your protocol

and submit it to this activity in Canvas. One submission will be enough for each group (should automatically show for both group members).

## Resources

- [1] RFC 7540 (<https://tools.ietf.org/html/rfc7540> ⇨ <https://tools.ietf.org/html/rfc7540>)
  - [2] RFC 7933 (<https://tools.ietf.org/html/rfc7933> ⇨ <https://tools.ietf.org/html/rfc7933>)
  - [3] RFC 6983 (<https://tools.ietf.org/html/rfc6983> ⇨ <https://tools.ietf.org/html/rfc6983>)
  - [4] <https://docs.python.org/3/howto/sockets.html> ⇨ <https://docs.python.org/3/howto/sockets.html>)
- Also covered in IS(3): UDP, IS(4): TCP, and IS(6): RAW
- [5] <https://docs.python.org/3/library/threading.html> ⇨ <https://docs.python.org/3/library/threading.html>)
  - [6] RFC 7231 (<https://tools.ietf.org/html/rfc7231.html> ⇨ <https://tools.ietf.org/html/rfc7231.html>)
  - [7] Switchyard: <https://github.com/jsommers/switchyard> ⇨ <https://github.com/jsommers/switchyard>)
  - [8] Web Proxy (Not a Standard - architecture in section 3.2.2 is intended):  
<https://www.ietf.org/rfc/rfc3040.txt> ⇨ <https://www.ietf.org/rfc/rfc3040.txt>)