159.352 Computer Work I:

Weight: 25%

This is a multi-part assessment with different intermediate deadlines.

Deadline: April 22nd 2020 (for part B)

PART A - Weekly tutorial exercises (5%)

Tutorial exercises reinforce the concepts introduced in lectures, and provide a foundation for this assessment. Therefore it is important to make steady progress and submit the tutorials on time.

Tutorial exercise	Due
Week 2	March 11
Week 3	March 18
Week 4	March 25
Week 5	April 1
Week 6	April 8

PART B – Distributed Social Network (20%)

Social media websites such as Facebook centrally store user data. Such web services are generally free for the users. Of course there is nothing such as free lunch. Companies monetize the user data with targeted advertising. Media reports[1] of data misuse has brought privacy concerns to the forefront. In this assignment we will explore the development of a simple distributed web application, which allows users store their own data and directly serve updates to their friends.

The conceptual framework of this web application is shown in Fig. 1. Each user runs her own HTTP server, which stores the respective user's data in a suitable file format (XML or JSON). The user data stored by each HTTP server include: **friend list**, **status file**, and **profile picture**.

A user's **friend list** file contains the friends' name and IP addresses of the friends' HTTP server.

The **status file** (e.g. status.xml) shows the particular user's history of status updates (<timestamp> <status text: max 200 characters> <likes: unique list of IP addresses>)

[1] https://www.theguardian.com/uk-news/2019/mar/17/cambridge-analytica-year-on-lesson-in-institutional-failure-christopher-wylie

The user and her friends can use any web browser. The web browser only communicates with the respective user's HTTP server, and never directly with the friends' HTTP server. The HTTP servers communicate with each over the network using API calls to exchange data. All the HTTP servers (user and her friends) will run the exact same code, which you will implement using raw sockets. You may use the server code in the tutorial exercises as a starting point. The objective of this assignment is to gain hands on experience with HTTP fundamentals, so **strictly do not use high-level frameworks** (e.g. Servlets, flask, node.js etc.,) since they abstract the low level HTTP functionality.

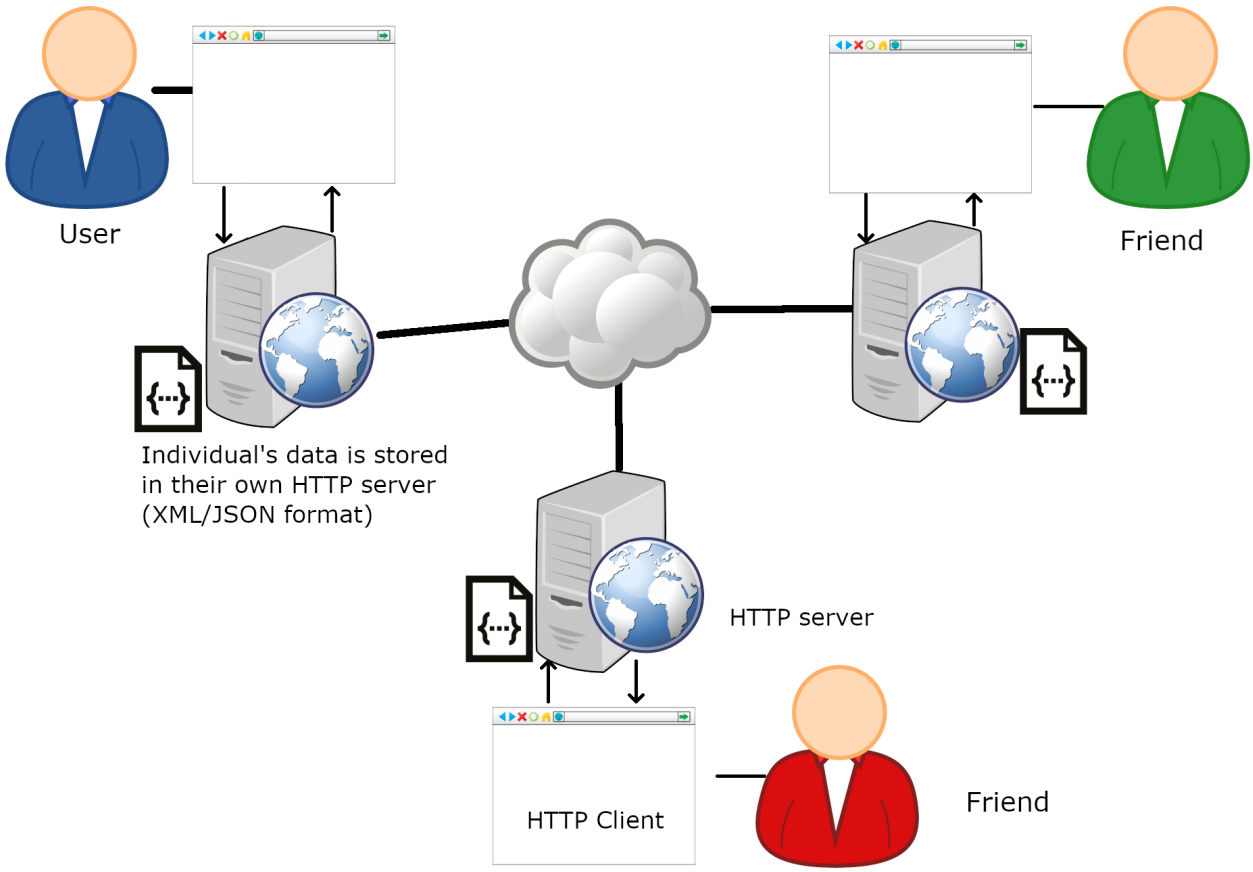

Fig.1 Conceptual framework

While there are many interesting features (e.g. poke, commenting) we could explore, we will keep things simple and limit it to the following functionality for the purposes of this assignment.

1. **Status update**

   First the HTTP server starts and listens on some chosen port number. Then, client connects to the server at this port and requests for the update page (update.html). You can create a simple page with a textbox, and 'Post' button. When the user submits the status message, it is sent to the server using HTTP POST method. The server then updates the user's status file (e.g. status.xml).

2. **Show friends**

   When a user points the browser to friends.html page on their HTTP server, the HTTP server iterates over the list of friends' HTTP servers, and requests for the friends' status file (status.xml), and picture. Here the connection is from the user's HTTP server to the friend's HTTP server. Technically the user's HTTP server is the 'client' and the friend's HTTP server is the 'server' in this connection. The friend's HTTP server will only grant access to the requested object iff the user (HTTP server) requesting the data is in it's own friend list. The user's HTTP server then dynamically creates a page showing each friend's name, picture, most recent status update, and the number of likes it received.

   As an optional exercise, you can improve response time by caching the friends.html file on the user's HTTP server. Using cache control headers, the user's HTTP server need not update information of friends who are offline (server not responding) or when there is no status change.

3. **Like**

   On the friends.html page, when a user click on a like button next to the friend's post, the browser sends this information to the user's HTTP server which then contacts the friend's HTTP server to update the 'like' field in the status file. The friend's HTTP server appends the user's IP address next to the status post.

Ideally you should be able to run and test your implementation on a real network. In case this is not practical then on a single computer you can create separate directory for each user and execute separate HTTP servers with different port number from those directories. In this approach you will need to record port numbers in the friend list, and status file.

Submission: A well-documented source code, complete with usage and test report via Stream site