CS 9668/4438 Internet Algorithmics
Assignment 4: Graduate and Undergraduate students
Due date: November 28

1. In the consensus problem that we studied in class, initially each processor p_i in the system has an integer value v_i and the goal is to ensure that at the end all processors select the same value v. Define the *half consensus* problem as above, except that now not all processors are required to select the same value, but at least half of them must select the same value. For example, assume that there are 4 processors p_1, p_2, p_3, and p_4. They initially have the values $v_1 = 2, v_2 = 6, v_3 = 1$, and $v_4 = 9$, respectively. In a feasible solution the processors might, for example, pick these values: p_1, and p_2 select value 2, p_3 selects value 6, and p_4 chooses value 1.

   Assume that at most f processors in the system can fail (for simplicity let us assume that f is even) and that processors might only exhibit stopping failures (i.e. when a processor fails it stops sending messages for the duration of the algorithm, but a processor might not be able finish sending all the messages that it must send in a round). Assume also that the network is complete, i.e., there is an edge between every pair of processors. Consider the following algorithm for the half consensus problem.

```
Algorithm half_consensus($id, f$)
Input: Processor's id and maximum number of processors that can fail
Output: Value selected by the processor to achieve half consensus.
$mssg \leftarrow \langle v_{id} \rangle$
rounds $\leftarrow 0$
$v_{\text{selected}} \leftarrow v_{id}$
loop {
        if $mssg \neq$ null then send $mssg$ to all neighbours.
        $mssg \leftarrow$ null
        for every message $\langle v \rangle$ received do if $v < v_{\text{selected}}$ then $v_{\text{selected}} \leftarrow v$
        rounds $\leftarrow$ rounds+1
        if rounds $= \frac{f}{2} + 1$ then return $v_{\text{selected}}$ else $mssg \leftarrow \langle v_{\text{selected}} \rangle$
}
```

   (*i*) (25 marks) Prove that the above algorithm correctly solves the half consensus problem in the presence of at most f crash failures, for any $f < n$.

   (*ii*) (5 marks) Compute the time complexity and communication complexity of this algorithm.

2. (5 marks) Consider a synchronous system in which processors can fail by *clean* crashes. This means that in a round, a processor either sends all its messages or none. After crashing, a processor does not send messages in the following rounds. What is the minimum number of rounds that the algorithm described in the previous question needs to perform to ensure that at the end all the processors select the same value? You need to prove that your answer is indeed the minimum number of required rounds and that all processors will achieve consensus.

3. (*i*) (30 marks) In class we studied the use of consistent hashing in Chord for finding keys in a peer-to-peer network. Assume that each processor p_i has a finger table that allows it to store its own address plus the addresses of two other processors. Which addresses must be stored in the table so as to, both, minimize the number of messages needed to find a key (or to decide that a key is not stored in the system), and to ensure that every key can be found?

You must explain your answer and show that the number of messages is minimized.

(*ii*) (5 marks) Prove that with your choice of fingers any key stored in the system can be found.

(*iii*) (5 marks) Compute the maximum number of messages that need to be sent to either find a key or to determine that the key is not stored in the system.

4. (*i*) (25 marks) Write a synchronous algorithm in Java that a processor in a peer-to-peer system can use to find a given set of keys, assuming that the consistent hashing scheme described in class is used to determine where to store the keys. Your algorithm **must** use finger tables to find the keys. To find a key your algorithm cannot simply send messages to the successors until the key is found. The finger table must be searched to find the processor closest to the position of the desired key; a message must then be sent to such a processor.

   To test your implementation we will use the simulator for distributed algorithms of Daniel Servos. For simplicity we will assume that only one processor, P, needs to find keys in the peer-to-peer system. Upon termination the algorithm must return a string of the form

   $$\text{"}k_1 : p_1 \;\; k_2 : p_2 \;\; \cdots \;\; k_r : p_r\text{"}$$

   where $k_1, k_2, \ldots, k_r$ are the keys that processor P needs to find and p_i is the processor storing key k_i for each $i = 1, 2, \ldots, r$. Before terminating, processor P will send an END message to its successor. An END message will pass from each node that receives it to its successor to ensure that all processors receive it. Processors other than P upon termination will return an empty string `""`.

   In class we discussed a simple search algorithm that does not use the finger tables, but in which a processor simply sends messages to its successor. A Java implementation of the algorithm will be posted in the course's website. You can use this algorithm as starting point for your solution. Since the simulator was not really designed to emulate peer-to-peer systems, in the network configuration file we will use the `DataKeys` command to specify the keys to store in the processors (these will be negative numbers), the keys that the processor P needs to find (these are positive numbers smaller than 1000), and the processor addresses in the finger table (these will be positive numbers larger than 1000). If this is a bit confusing, look at the Java code posted in the website to see how the two above sets of keys and the finger table of a processor can be determined from the configuration file.

   If you cannot write a program in Java, you will be allowed to submit a solution in pseudocode. However, you are strongly encouraged to write your algorithm in Java.

5. (5 marks) **Optional question.** You can download from the course's website a chat client and chat server that allow messages to be exchanged between a client and a server running on different computers. There are 2 versions of these programs, one written in C and the other written in Java.

   Study the above programs and modify them to write either in C or in Java a file server and a client. Your file server will be executed first and it must wait for requests from clients. After a client establishes a connection with the server the user sitting at the client's computer types the name of a file and sends it to the server. When the server receives the name of the file it checks that the file is stored in the computer hosting the server. If the file is found a FOUND message is sent to the client followed by a sequence of messages, each containing one line of text from the file. After the file has been transmitted and END message is sent to the client. If the file is not found in the server an END message is sent to the client.

   The client saves the file and it closes the connection when it receives an END message. For simplicity you can assume that all the files are text files and all of them are stored in the same directory or folder where the server program is stored.

You need to submit your code through OWL. You can write the program in another programming language if you want, as long as we can run your program on the research network.