Homework 4 (Due: 12/3)

1. Chat Application —aka— Fun with Sockets

This question asks you to implement a multicast chat application in which each participant can send and receive chat messages from all other participants. We will test the interoperability of this application in class. The aim of this question is twofold: i) to give you a basic learning experience with sockets, the widely used API through which TCP/UDP is used by most distributed applications; and ii) to give you a sense of the sometimes subtle challenges of interoperability.

As before, we require that you implement your program in Java and recommend that you do this part of the homework with one (1) partner with who you may collaborate freely – don't get wedged. There are many online introductions to socket programming, and you should look at one first if you have not used sockets before, e.g., http://download.oracle.com/javase/tutorial/networking/datagrams/index.html

The mchat Application: It should run as follows. When the program is started, it should join the group of nearby participants. It should then let the user enter a message and reliably deliver the message to all of the participants in the group, including itself. Each participant that receives a message should display it. The GUI is up to you, but note that there are no extra points for a fancy GUI.

The mchat Protocol: It is very simple, probably good enough for our purpose but not very good! The protocol runs on UDP using a configurable port and IP multicast address to define the group. There are four messages you may use to have your application join the group and reliably send and receive chat messages. Each message is sent as a single UDP message with ASCII contents:

- "GDAY *nickname*" where *nickname* is a username with only alphanumeric characters and no spaces. An application periodically sends this message to the group IP multicast address and port to announce to all the other participants that it is a member of the group. This message is not sent reliably. An application that has received a GDAY from a participant within the last 30 seconds considers that participant to be part of the multicast chat group. Chat messages for the participant should be delivered to the source IP address and port of the GDAY message.
- "GBYE *nickname*", which an application sends as a courtesy, again to all the other participants via the group IP multicast address and port. This message tells them that the participant with the source IP address and port of the message is leaving the chat session. This message is not sent reliably.
- "SAYS *nickname seq-number message*" where *nickname* is the username of the sender, *seq-number* is an integer sequence number that is larger for each new message and *message* is the string that is being sent from the sender to the receiver who is one participant in the multicast chat session. It is your job to deliver this message reliably using timeouts, retransmissions and acknowledgements (the YEAH message).
- "YEAH *seq-number*" where *seq-number* is an integer sequence number taken from a SAYS message. This message serves as an acknowledgement to the receiver that the sender has received a SAYS message from the receiver with the corresponding sequence number. It is sent to the

source IP address and port of the corresponding SAYS message. YEAH is used to implement the reliable delivery of SAYS messages.

Your job is to build the application using UDP sockets and only the four message types. Here are a few issues to think about:

- The GDAY and GBYE messages are sent from one sender to all receivers with one multicast network transmission but the SAYS and YEAH messages are typically sent from one sender to one receiver. Thus to multicast chat messages your application must send a copy of one SAYS message to each participant in the group.
- Since you can only use UDP and it isn't reliable you will need to implement timers, retransmissions and acknowledgements to reliably deliver SAYS messages. You should implement only a simple scheme. We recommend stop-and-wait with an incrementing sequence number.
- Think about the structure of your application before you start coding. It is likely that you will have multiple threads of control in your application (though you can use events if you prefer). Choose a simple structure and do not worry about performance.
- Think about how someone else might interpret the above differently and try to program to send clean output and tolerate various input. We recommend that you include code to print out network messages you receive but cannot interpret properly too.
- To test, you would ideally want to run a few instances of your application on one machine. Unfortunately only one process can bind to a given port at a time. You can test with two machines simply enough. Alternatively, you might modify your application to bind to one port but send GDAYs to a range of, say, 5 ports. This would let you run five instances of the app on one machine, with each instance binding to a different port and sending GDAYs to the range of all 5 ports ensuring that the instances find each other. No changes to SAYS/YEAH code would be needed. This is only a suggestion to help with testing; if it does not make sense to you now simply ignore it.

Questions and Turn-in: You get the bulk of the points for this question simply by writing a working app.

a) Turn in your code. We want to see what you wrote and if your program works. Also bring your app to class. Let's see how many people can connect!
b) As the size of the group increases the load on the network will increase, eventually causing problems and limiting how large the group can be. Describe one aspect of the increase with N members as N grows and one implementation strategy that would be effective at reducing the problems associated with load.
c) A problem in the design above is that it does not ensure that every member of the group sees the messages in the same order that they occur. This is surprisingly difficult to guarantee. For example, suppose each node timestamps chat messages as they are received and displays them in timestamp order. Because it takes a finite amount of time to send the message to the entire set of members it is possible that members A and B receive two different messages that are sent close in time in a different order. We can have the sender timestamp the messages instead so that every participant gets the same timestamp for a message. However, the clocks on different computers are not exactly

the same, so now A might send before B and all nodes might print the messages in the reverse order if A's clock is running slow. Luckily a friend suggests that you should read "Time, Clocks, and the Ordering of Events in a Distributed System" by Lamport. (This is a classic paper, fun read, and idea you should know about. Enjoy!) Your job is to give a brief description (1 para) of how we can solve the problem using only the message formats we already have and the ideas in the paper.

2. Digging in the DNS

Your task is to use the DIG tool to explore the DNS. DIG is installed on the CSE lab machines and is widely available with documentation online. Your friend in Perth, Australia wants to send mail to you at pmp-star@cs.washington.edu, which requires that they find the hostname and IP address of a machine that accepts mail for accounts at cs.washington.edu. Assume that your friend knows only the root DNS machine l.root-servers.net to start, and that when multiple nameservers are returned in response to queries the one with the highest IPv4 address is selected. Further assume that there is no caching of DNS responses anywhere, and that no packets are lost. Draw a diagram similar to Fig 7-6 in your text that shows the series of queries and responses. Give domain names and IP addresses.

3. Design Exercises

a) You buy a wireless AP for your home from BestBuy to hook up to your DSL line. On the back of the box it says "QOS support for VoIP – voice calls are carried with low delay". By this it means that the AP looks at the IP Traffic Class bits and puts packets marked for Expedited Forwarding (with Differentiated Services Code Point (DSCP) values of 46) into a weighted fair queue with a high weight compared to the other traffic. Your goal is Skype calls that work well even when you run other applications on your home network at the same time. List all the assumptions you must make in order for the AP to meet your goal.

b) The DNS is a distributed system that is designed to meet several goals:

   i. Continues to work well as the size of the overall system grows large
   ii. Provides highly available service to clients
   iii. Provides rapid name resolution to clients
   iv. Serves many different organizations

   Make a table that lists the specific features in the design (or implementation) of the DNS that help it to meet these goals.

4. Textbook

Questions 5.17, 5.19, 6.12, 7.7

□□□