CS 162 – Fall 2011 TA: Angela Juang
Discussion Notes Week #05

CS162 Discussion #05: Networking

Announcements

- Project 1 initial design + individual portion due TODAY, 09/27 @11:59pm
    - Sign up for design review slots 09/28 and 09/29 at http://goo.gl/KmRYu
    - Give Angela a hardcopy of your design doc!
    - Only one person in your group should submit the initial design
    - Everyone must be present at the design review

Moving toward The Internets

We've talked about how to make multiple processes and threads run together on the same machine, but what about having different machines communicate with each other? For different machines to be able to interact with each other, they need to be able to send information over a network. This requires that we have a way to get data through a network of links and routers from a source to a destination. We can establish a connection between two endpoints using several different strategies, including:

- *Circuit switching.* A circuit-switched network must reserve a path between the source and destination for each flow of data for the duration of the connection. Once a connection is established, we can continuously send a stream of data across the link without worrying about what other connections are there.
- *Packet switching.* A packet-switched network dynamically allocates resources based on what packets (chunks of data) are being transmitted. Data must be broken up into packets, each of which may possibly take a different route even if they are part of the same flow.

These two schemes illustrate ways that we can choose to have data transmitted over the network. These basic concepts make sense for a single connection in which we only have to worry about one source and one destination. However, we obviously don't want to limit the network to only be able to handle one connection at once, so we have to keep some additional considerations in mind. How do we handle sharing the network between an arbitrary number of connections? Both circuit-switched and packet-switched networks have different ways of dealing with this situation:

- *Circuit-switched: Time Division Multiple Access (TDMA).* First of all, since circuit-switched networks require us to reserve the entire route beforehand, if there are not enough resources to allocate an entire path from source to destination, the network is considered busy and a connection cannot be established. Once a connection has been established, we can share a channel with other connections on the same channel by using time division multiple access (TDMA). In TDMA, time is divided up into frames, which is further divided up into slots. Each flow is allocated one slot and may send data in that one slot per frame, and may not send data in any other flow's slot even if there are unused slots in a particular frame. This is because if one flow is allowed to use another flow's slot, the routers and endpoints will have no way to tell which flow the data actually corresponds to. Due to this reason, in this scheme, we can end up with a lot of wasted space in the pipeline if some flows are relatively quiet and a few flows are extremely busy.
- *Packet-switched: Statistical Multiplexing.* Packet-switched networks can continue accepting new connections without creating a busy signal, but the output will decrease as the links get more and more congested. These types of networks rely on a scheme called statistical multiplexing, in which all

connections share the entire link. This way, the entire link is available to all flows, and busier flows can adjust to use more of the resources while quiet flows can continue using less. We make the reasonable assumption that on average, connections should have burst periods (periods of high activity) at different times, so each flow should be able to take advantage of the shared link space when they need it.

Note: In your next project, you'll have to do some network programming in Java. It might be helpful to take a look at the following classes to get a feel for how to work with Java sockets:

- `java.net.Socket`
- `java.net.ServerSocket`
- `java.io.DataOutputStream`

I accidentally the whole project

Now that you've had a taste of working with your group for CS 162, you might have seen that coordinating a group of 4-5 people can be very challenging. As you go into the implementation phase of the project, here are a few tips to help you make sure that things go smoothly.

Have organized group meetings, and check up on all members regularly. It's important to communicate with your group early and often, so nobody gets left behind or forgets what they're supposed to be doing. If you don't know a member's progress on their part of the project, you should assume they haven't done it. It can be hard to coordinate frequent in-person meetings, but keep communication flowing through email, chat, or even a bug tracking system (though that might be overkill for this class). When you do get a chance to meet in person, make sure your meetings are actually productive and efficient. A great strategy is to prepare an agenda on a Google doc or other shared document before the meeting, and give everyone in the group a chance to add topics to it. When the meeting comes around, you'll have a list of things to focus on and won't forget important things to address. In addition, make sure all decisions, whether made individually or as a group, are documented in writing somewhere so there's no confusion on what was actually decided upon.

If small issues come up, don't be afraid to bring it up. It might be uncomfortable to be the one to bring up issues within your group, but catching problems early and trying to fix them as soon as they arise will make it easier to change – the worst is if you continue to wait things out and end up having an angry explosion at the end of the semester when it's too late to do anything about it. As soon as you feel uncomfortable with anything in your group or are having trouble handling the workload, let your TA know immediately! We're here to help and advise you on ways to alleviate the situation, and if necessary, we will take steps to intervene with problematic group situations so that they don't get too out of hand.

Have good coding etiquette and iterate often. It's good practice to make sure that you're always checking in working, well-commented code, preferably that has also been tested already. You should think about using some kind of testing framework to test your code (e.g., JUnit and EasyMock). We recommend that you do many fast iterations of the design-implement-test cycle throughout the implementation phase. This way, you can constantly reevaluate and improve your approach to various parts of the project after having had some experience with implementation and testing with your previous iterations.