Senior Project Reflection

Andrew Fisher

My senior project involved a lot of challenge due to changing the focus radically in the middle of the semester. It started as a project to simulate both client-server and peer-to-peer networks as a comparison, to look at metrics such as reliability, scalability, and speed. I was hoping to find a simulation framework that natively supported both traditional client-server delivery protocols (HTTP(S)/FTP) and the more well-known P2P overlay delivery protocols (Gnutella, BitTorrent) so that the differences in environment were minimal and the comparison would have meaning. As it turns out, due to differences in how the various application-layer protocols interact with the transport-layer protocols (TCP/UDP), there are no simulators that natively support both paradigms. There exist simulators for P2P, simulators for CS, and simulation frameworks with free topology definition, but none that overlap all of these categories. This meant that my original project could not be done as I originally hoped, as designing my own simulations for existing protocols in a free topology framework would be far outside the scale of the semester.

This was the major setback on the road this semester, since I had to change the focus of my project to instead another problem in the P2P world: how to efficiently and reliably find resources on a network. This is known as the Search Problem and is an outstanding question due to the difficulty in discovering which nodes on the network are carrying the resource one is looking for. Due to decentralization, this is much more difficult than on a centralized server-based network, where the resource is known to either exist on the server or not exist at all.

The main thing I learned over the course of this project was how P2P networks implement searching and how they deal with inefficiencies and problems of reliability, especially when the network starts to become larger. In particular, I learned that, at least early on, networks often used incredibly

simple algorithms (breadth-first search, random walk, etc.) that do not require a lot of overhead, but trade off efficiency, bandwidth, or reliability. With this in mind, I took the simplest of these algorithms, Random Walk, and decided to write a simulation for it by modifying a routing simulation in the OMNet++ network simulation framework.

OMNet++ is a C++ based framework that falls into the "free topology builder" category of network simulators. Pre-built as part of the OMNet++ IDE (a reskinned Eclipse specifically for simulation deployment) is a random routing simulation that generates packets and moves them around the network. I took this as a base for my simulation since a Random Walk search is mechanically similar, and created my own SearchNode and SearcherApp classes in the style of the existing Node and App classes to include the logic and metrics for the search algorithm. My algorithm essentially picks a random destination, routes a packet, checks if the destination has a Boolean that defines a file, and then either routes back to the original sender or continues randomly choosing destinations to search. This approach was relatively simple, but still had its own setbacks.

Challenges faced in this project largely fell into three categories: NED problems, C++ problems, and random number problems. OMNet++ uses a proprietary language called NED, which I believe to be a JSON framework of some kind, to define parameters and connections for modules/class objects. Unfortunately, NED barely supports data structures and is essentially built to handle only primitives like String, int, and so on. This would be fine, except that the designers of this routing simulation decided to store the entire list of destination addresses as a hardcoded string in NED, which was a real challenge to change to be a dynamically modifiable list of destination addresses for search to look at. NED in general created more problems than it solved due to the small amount of support for higher-level features found in C++.

The C++ problems largely had to do with addressing and pointers, as is the case with most C++ projects. Specifically, some things needed to be initialized when the NED module was instantiated, while other needed to be modified and destroyed during the simulation loop. Figuring out all the right pointing to make sure those changes could be made was a challenge, but also solidified my understanding of the C++ object model. Additionally, figuring out what needed to go into which methods to ensure that the right things were (or were not) being updated on every simulation tick required thinking inductively, since I had to determine how to write a specific class that would be applied to a generalized set of nodes, and would be repeated multiple times per second. Figuring out what the base case (initialization) and inductive step (packet handler method) had to be to ensure everything talked nicely was a challenge of my discrete math skills.

Finally, the random numbers by default in OMNet++ are pseudo-random to allow repeat trials. Additionally, they are pulled from a user-specified distribution that has quite a few options. To figure out what I needed in particular, I learned about what "random" numbers mean for a computer and how to generate a more random number in OMNet++, from a useful distribution for my project.

Overall, I am proud to say that I created a mostly working Random Walk search simulation, and am especially happy that it works for most cases (the case where a set of nodes in a subgraph all don't have the file is an infinite loop due to undecidability), given that I started this project 5 weeks late compared to other seniors since my original project did not pan out. It taught me a lot about networking, C++, and discrete event simulation, all of which will be useful skills at some point I am sure.