

22C: 196: 001 Peer-to-peer Networks

List of projects

Total points = 100 (worth 30% of your grades)

Assigned February 19, 2009, due May 7, 2009

This is a first draft of the projects. Pick a project topic, form a team of two persons, and start working. At the end, you need to write a report, and arrange a demonstration of your work if appropriate. The length of the report should be similar to that of a conference paper (10-15 pages), and it must document all your activities, along with a list of relevant references. Do not include codes, but you can put them on your webpage, with a citation containing the URL of the link.

Project 1

You have to implement a P2P network using the open source software BRITE that can be downloaded from

<http://www.cs.bu.edu/brite/download.html>

The simulator reads a *topology* file containing the physical nodes, randomly selects a portion of the physical nodes as the overlay nodes, and then generates a P2P network on top of the topology specified in the topology file. The number of overlay nodes should be configurable by the user. Use BRITE to generate the topology of the physical layer and output topology of the P2P overlay as a topology file. After the P2P network is built, the system should be able to output the following information:

- The number of nodes and edges in the P2P network.
- The ID, node degree and coordinates of any node in the P2P network.
- The 1-hop neighbor nodes of any node in the P2P network.
- The overlay route and the underlying physical route between any two nodes in the P2P network.

The simulator has the following functionality:

(a) It can accept an input file, and places it (deterministically or randomly) on one or more nodes in the P2P network.

(b) It can issue a *query message* from any node to search a file in the P2P network. The query is configured with a TTL value. The query is broadcast to all neighbors. When a node receives a query, it searches the local "file list" file. If there is a match, then it generates and sends a *hit message* back to the source node that issued the query. If the file cannot be found in the local file

list, then the query is *forwarded* to all the neighbors except the one sending the query. The hit message is sent back via *reverse path routing*. To make it possible, each node maintains a routing table that maps the message ID to its sender.

Try out the simulator and its functionalities. Issue a query from any node in the network and retrieve a file. No specific interface is necessary. You may implement it either with GUI or with simple text modes.

Now, generate graphs of size up to $n \geq 5000$ nodes. Arrange to randomly place copies of 20 files – each file should be placed at 0.1% to 1% of the peers. Now, run various search algorithms (a) using Gnutella-type flooding (b) using the random walker model. In each case, compute the search time from multiple trials, as well as the total number of nodes visited to locate the object. Then compare the performances of the two search algorithms. Add explanations about your findings.

The project report should be (a) a user manual of your simulator, introducing how your project compiles and runs. All the functionalities described above should be implemented in your project and introduced in your project report. (b) The source code - remember to provide a Makefile to automatically build your project. (c) Executable files of your project. Note that your project must be self-contained. So please include all the files necessary for your project to work well.

Project 2

Implement various skip graphs with 10,000 - 1,000,000 nodes, and do the following experiments:

(a) Implement the protocols for key insertion and key deletion. Plot the time (i.e. the number of steps) needed to perform these tasks against the size of the graph.

(b) Assume an arbitrary node as the hotspot and consider 10,000 accesses from randomly chosen nodes across the network. Compute the load distribution around a hotspot in the skip graph and verify if it agrees with the theoretical result.

(c) Induce random failures, and study how searches are affected by such failures (i.e. find out what percentage of them are failed searches). Also, find out how many random failures are likely to partition the skip graph.

Project 3

PeerSim (<http://peersim.sourceforge.net/>) is a free, multi-threaded, discrete event simulator to evaluate, investigate, and explore peer-to-peer protocols. PeerSim runs in several UNIX-like operating systems. Its goal is to make the understanding peer-to-peer protocol source code easy, to facilitate easy comparison of different protocols, and to have a reasonable performance for the simulation runs. PeerSim already supports Chord, Accordion, Koorde, Kelips, Tapestry, and Kademia. These implementations are specific to PeerSim. They consist of substantially fewer lines of code than the real implementations.

Use PeerSim to implement a Chord network with a key space of size $n=2^{16}$, and place 1024 nodes (machines) on the network. Name each machine with a string of three randomly chosen alphabets 'a'-'z'. Name 256 objects each with a string of three randomly chosen hex digits (0-F) and place them on the different machines. Allow $2 \log_2 n$ nodes for each node to support bidirectional routing. Simulate the *search object* operation for 50 different accesses originating from random machines a few designated objects, and verify that the search indeed terminated at the host machine. Tabulate these operations and in each case, record the number of steps needed to perform the search.

Simulate the addition and deletion operations of nodes. Run the stabilization protocol after every 10 addition and deletion operations, and record partial snapshots to verify that the new nodes were indeed correctly added and the departing nodes were flushed out of the system. Write a report explaining how to use PeerSim, present your results, and analyze them.

Project 4

(This project is for undergraduate students only. Expertise in Python is essential) Professor Ed Felten wrote TinyP2P using 15 lines of Python code to demonstrate how simple it can be to write such an application. It works for small networks. The code for TinyP2P can be found in

<http://python.about.com/b/2007/09/18/the-worlds-tiniest-p2p-program.htm>

(The earlier documentation in <http://freedom-to-tinker.com/tinyp2p.html> disappeared)

Study the implementation of TinyP2P. Use the ideas to implement a P2P network that can accommodate up to 64 peers. At the end, each peer should be able to store up to eight objects in

the local memory, and access any of the objects of any peer efficiently. At the end, you have to write a report, submit the source code, and demonstrate a working system.

Project 5

Churn in a P2P system is a major problem that affects its performance. Investigate why churns are caused, search for experimental data on churns.

Two techniques to minimize the effect on Churn are object replication (fairly well-known and easily understood) and randomized routing (needs to be studied, but think about what the motivation can be). You need to do the following:

(1) Implement a Chord network with 1024 nodes (Feel free to use PeerSim introduced in Project 3) that supports unidirectional routing. Store k copies of each object at various nodes (how will you figure out where to store the replicas?). Simulate the role of an adversary and report if you noticed any improvement in the resilience (as measured by the number of failed searches) for different values of k

(2) Explore if the use of randomization will help in expediting search in a Chord network in presence of churn. Suggest a possible scheme for adding randomization to the search or routing algorithm to guarantee better resilience. You have to simulate the role of an adversary.

To figure out the resilience of the network, consider both adversarial churns and random churns.

Project 6

Create small world graphs with $n \times n$ nodes ($10,000 \leq n \leq 100,000$). Use Kleinberg's model to generate the topology of the network: assume a two-dimensional lattice, each node is connected to its four neighbors, and there exists **one** long distance link connecting a remote neighbor. The probability of the existence of a long distance neighbor from u to v will be proportional to $\text{distance}(u,v)^{-r}$

(1) Let $r = 0$. Use a greedy algorithm to route a query to a (lattice) distance of at least 64 different randomly chosen nodes. Calculate how many steps it took in each case.

(2) Repeat the experiment with $r=2$

(3) Repeat the experiment with $r=3$

Does your experiment support Kleinberg's theory?

Project 7

Exploring Space-filling Curves

In a large distributed system, finding a node at or near a specific location can be hard -- especially if you want to do it efficiently. Several research projects (including the Hourglass project at Harvard) have begun approaching this problem in the following way: a structured overlay (or DHT) provides a one-dimensional space; let us imagine that locations (coordinates) are one-dimensional and also let us assume that each node knows its own location; nodes can "post" their locations into the DHT; a query for a node at or near a coordinate would perform an "expanding" query centered at this spot on the DHT; walking away from this spot in both directions along the ring, it will soon find the nearest node, and we are done. Now, how can this be achieved if we drop the assumption that coordinates are one-dimensional? Answer: use a *space-filling curve* to map multi-dimensional coordinates to one-dimensional coordinates with the hope that points that are nearby each other in one space are nearby each other in the other. The Hourglass project currently uses a Hilbert space-filling curve to do this and has found it works reasonably well.

While these one-dimensional coordinates could be stored in a DHT and queried using an expanding search, such an infrastructure does not exist. The goal of this project would be to take an existing DHT like Chord, and add on this search capability and evaluate its effectiveness.

(The project specification is currently incomplete. If you are interested, then stop by and we will discuss about what needs to be done)