CSE 498N/598N HWP 1: Unstructured Peer to Peer Networks

Assigned: Tuesday, Jan 14
Due: Tuesday, Feb 11, 12:30PM
(INDIVIDUAL EFFORT. LATE SUBMISSIONS WILL NOT BE ACCEPTED)

In this project, we will implement a peer to peer scheme similar to Gnutella. The system consists of a number of independent peers, each of which serve some files. These peers listen on a TCP port and maintain simple *key:value* tuples. In the case of GNUtella, this could be *Key=BritneySpears:Value=MP3song*. The functionality that we will implement is the ability to search and serve the values (given a certain key). There are a number of components necessary to implement this project.

1 Peers

The first problem that we need to solve is to name and identify the peers. Each peer independently chooses its name. For this project, we will use *node IP:port* as the name of each peer. Each peer will listen on this port and provide service to other peers (answering file queries, service the files etc.) on this port; you choose this specific port. You could run a number of peers on the same machine by choosing different port numbers for each one of them.

The peers will maintain *key:value* tuples. The peers will provide the following interface for services (note that the services are described in a 'C' like pseudo function call. You are free to implement it in a fashion that is convenient for you):

- **get(key)** This service will return the *value* associated with a given *key*. The key should be among the keys listed in the *list* service.
- **set(key, value)** This service will associate the *value* with the *key*. Existing values are overwritten with the new contents.
- **list()** This service will list all the keys that are available at the peer (set using earlier *set* operations).

1.1 Sample output:

We illustrate a sample interaction with a particular peer in wizard.cse.nd.edu:6003. Your interactions are illustrated in `typewriter font`.

```
$ telnet wizard.cse.nd.edu 6003
SET PinkFloydWall SongDataWillGoHere
OK SET
LIST
OK LIST
PinkFloydWall
GET PinkFloydWall
OK GET
SongDataWillGoHere
```

2 Location service

Next, these peers will maintain information about other peers that are currently online. In general, there is a limit to the number of peers that you can maintain information about. For this project, the peers will be restricted to manage location information about **two** other peers. For debugging purposes, the peers will continue to print location information such as peers entering and leaving the system.

Peers identify each other by exchanging the *identification_t* structure. For this project, you will exchange the name, the Internet port number and Internet address where you can be reached in the *identification_t* structure. Peers can also maintain the round trip times to the different peers in order to choose the "best" peers. You can refer to the COMPUTER NETWORKS book by *W. Richard Stevens* [1] for sample code on using network system calls. The sample code from this book is available online at http://www.kohala.com/start/unpv12e.html.

```
typedef struct identification {
  char name[32];               /* Name of the current client */

  // Specify how we can be contacted.
  in_addr_t  location;       /* IP address of the client */
  in_port_t  port;           /* port where the client is listening */
} identification_t;
```

There are a number of different ways to locate other peers. For this project, you will use peer-to-peer techniques to locate other peers (and not a centralized approach). The peers will directly locate other peers without any centralized data structures. Peers can utilize multicasting (all peers listen on different multicast channels) or broadcasting to identify other peers. Peers can broadcast a query asking other peers to identify themselves or new peers can initially broadcast their identity in order to join the community. You should be able to locate instances of your own peer running on different hosts (you would have to explicitly start a number of peers). You may also be able to locate peers developed by your classmates.

2.1 Sample output:

This is a sample run for how you might print other peers entering and leaving the system.

1/9/2003 10:30 'John Doe' ENTER darwin.cc.nd.edu:6780 20 msec

1/9/2003 10:35 'John Doe' MAINTAIN darwin.cc.nd.edu:6780 30 msec

1/9/2003 10:36 'John Doe' LEAVE darwin.cc.nd.edu.edu:6780

1/9/2003 10:40 'Jane Doe' ENTER wizard.cse.nd.edu:6003 100 msec

3 Service the tuples: Query routing

The next step is to be able to access *key:values* that are available in peers that may not be directly known to the current peer. When contacting remote peers, you are only allowed to directly contact a peer that you already know about. All other peers should be queried through peers that you know about. For example, suppose we know about peer FOO on host1:port1 and we get a request for BAR on host1:port1, we have to send the request to FOO which might forward the request to BAR. The peers will provide these services:

- **search(searchKey, hopCount)** If the requested key *searchKey* is available in the peer, the identity of the peer (in a printable *hostIP:port* format) is sent back. If the key *searchKey* was not available, a recursive **searchget** is invoked by this peer (on behalf of the requestor) on all the peers that it knows of (restricted to two for this project). Every such forwarding decrements the hopCount. Once the hopCount reaches 0 without successfully finding the file, the system returns an error message.

Note that your implementation might return multiple values for the same key. It might also return an KeyNotFound error, even though there is a path available from the source to the destination.

- **rget(peerHost, peerPort, key)** This service will return the *value* associated with a given *key* in the peer at peerHost:peerPort. You are only allowed to directly contact the peer at peerHost:peerPort if your own internal routing tables know about this peer. If you do not have direct knowledge of the peer, you should forward it to a peer that you know.

4 Submission

You are free to choose your program implementation language. In general, choose a language that you know and are comfortable with. Submit your project, along with a succinct report called REPORT.txt (plain text is fine) describing your approach, the merits of your approach and compilation instructions. You will turn in your complete project as a single tar file. You can use the course AFS dropbox at
/afs/nd.edu/coursesp.03/cse/cse498n.01/dropbox/$< loginid >$.
Evaluate your implementation on the following issues in the REPORT.txt:

1. **interoperability:** How does your peer recognize your friend in a different host/operating system. For example, if you are working in wizard [Sun Sparc machine running Solaris], can you identify your friend in the dorm using a Pentium III running Linux?
2. **consistency:** How quickly do peers realize when a peer crashes so that the display that you get accurately reflects the peers that are currently online?
3. **scalability:** If your peer suddenly becomes popular because of the files that your provide, how much load can you tolerate before your system crashes? Does your service degrade gracefully? Are you immune to denial-of-service attacks? (wherein, peers repeatedly open connections to you to prevent you from servicing other, legitimate users)
4. **robustness:** How reliable is your system against failures? Does your peer recognize forwarding loops? (wherein the requests are forwarded around in a loop without making progress towards the destination)

References

[1] W. Richard Stevens. *UNIX Network Programming: Networking APIs: Sockets and XTI*, volume 1 of *ISBN 0-13-490012-X*. Prentice Hall, 2 edition, 1998. Sample code from this book is available at http://www.kohala.com/start/unpv12e.html.