CSCI 4770/6770 HW 1: Locating Peer Beacons

Assigned: Tuesday, Jan 8
Due: Tuesday, Feb 5, 11:00AM
(LATE SUBMISSIONS WILL NOT BE ACCEPTED)

1 Motivation

Locating computing resources is an important first step in accessing ubiquitous resources. For example, when you walk into this class, you want to know who/what else is accessible. You can use this information to communicate with your "friends", find the closest printer etc. You can also use this information to allow your "friends" to communicate with you. For ubiquitous computing, location management is part of a larger problem of authentication, authorization, protocol negotiation, service management etc.

In this project, we will implement a simple scheme for locating other beacons that are currently online. Beacons are defined as software representations of physical entities. Beacons can represent printers, scanners, LCD projectors, people, microwave ovens, etc. Depending on the object that a beacon is attached to, beacons can provide any number of different services. For example, a beacon attached to a printer can advertise the printers capabilities (postscript/PCL, color/grayscale, 1200 dpi, inkjet etc), accept jobs for printing and provide status information (printer jam, out of paper, out of ink etc.). A more complex beacon can make requests to other beacons to accomplish its tasks. For example, the printer beacon can in turn contact other beacons to convert a powerpoint document to postscript format so that it can be printed. The printer beacon can also contact a clearinghouse to charge the user for the printing costs.

In general, there are a number of different ways for beacons to identify other beacons. Some popular techniques are:

1. **Central Server:** This approach uses a centralized server. Every beacon registers itself with this server. One would query this central server to search for other beacons. You have to first know where this central server is located before you can ask it for the beacon locations (boot-strap problem). Some popular examples utilizing this approach include napster (www.napster.com), AOL Instant messenger etc.

2. **Peer-to-peer:** Here the beacons directly locate other beacons without any centralized data structures. Beacons can utilize multicasting (all beacons listen on different multicast channels) or broadcasting to identify other beacons. Beacons can broadcast a query asking other beacons to identify themselves or new beacons can initially broadcast their identity in order to join the community. Gnutella (www.gnutella.org) utilizes such peer-to-peer techniques.

2 Project Description

There are two components to this project; first you will develop beacons (that are standalone peers) that listen in on a TCP port and maintain simple *key:value* tuples. Next, these beacons will maintain information about other beacons that are currently online and print the results on the terminal. For debugging purposes, the beacons will continue to print location information such as beacons entering and leaving the system.

2.1 Beacons

The beacons will maintain *key:value* tuples. We will not worry about how this meta-information is utilized. The beacons will listen in on a TCP port (of your choice). The beacons 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):

- **open(passwd)** All the requests to a beacon should be preceded by the open function. Beacons need a way to authenticate the user who is making a particular request. For our project, we assume that any *passwd* is always valid. The beacon will return a token that identifies a particular session. Further requests to this beacon should be accompanied with this token for service. A request without a valid token should be denied.
- **get(token, key)** This service will send the *value* associated with a given *key*. The key should be among the keys listed in the *list* service. Requests for a key that is not available should be denied.
- **set(token, key, value)** This service will associate the *value* with the *key*. Existing values are overwritten with the new contents.
- **list(token)** This service will list all the keys that are available at the beacon using earlier *set* operations.
- **close(token)** This service will end the session with this particular beacon.

2.2 Sample Output:

We illustrate a sample interaction with a particular beacon in greenhouse.cs.uga.edu:6003. Your interactions are illustrated in `typewriter font`.

```
% telnet greenhouse.cs.uga.edu 6003
OPEN surendar
OK token0
SET token0 PinkFloydWall SongDataWillGoHere
OK SET
LIST token0
OK LIST
PinkFloydWall
GET token0 PinkFloydWall
OK GET
SongDataWillGoHere
CLOSE token0
OK CLOSE
Connection closed by foreign host.
```

2.3 Location service

For this part, your goal is to identify other beacons which are online at the same time. Beacons identify each other by exchanging the *identification_t* structure. For this project, you will exchange your name, the Internet port number and Internet address where you can be reached in the *identification_t* structure. The beacons print this information without any explicit user interaction. You can refer to the COMPUTER NETWORKS book by *W. Richard Stevens* 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.

```
// Structure that is exchanged between clients to help identify the
// client location, how to contact the client and what protocol to speak
typedef struct identification {
  // Identify who we are
  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 */

  // Specify my credentials
  char key[32];                /* My authentication key. Not used for
                                  this project  */

  // Specify how to talk to us. Not used for this project
  unsigned int  type;          /* whether we talk XML, HTTP, Corba protocols */
  unsigned int length;         /* Length of protocol specific data in buf */
  void *buf;                   /* Protocol specific buffer */
} identification_t;
```

For this project, **you will use peer-to-peer techniques to locate other peers (and not centralized approach)**. You should be able to locate instances of your own beacon running on different hosts (you would have to explicitly start a number of beacons). You may also be able to locate beacons developed by your classmates.

2.4 Sample output:

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

1/9/2002 10:30 'John Doe' ENTER gemini.cs.uga.edu:6780

1/9/2002 10:35 'John Doe' LEAVE gemini.cs.uga.edu:6780

1/9/2002 10:40 'Jane Doe' ENTER greenhouse.cs.uga.edu:6003

3 Submission

Please 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. On gemini, please use `/home/profs/surendar/bin/turnin UBICOMP HWP1 <your tar file>` to submit your assignment. You can submit your assignment multiple times. I will only use the latest submission. To see the files that you had submitted, try `turnin UBICOMP HWP1`. **Remember, I will randomly choose students who will be asked to explain their approach in person**.

Evaluate your implementation on the following issues in the REPORT.txt:

1. **interoperability:** How does your beacon recognize your friend in a different host/operating system. For example, if you are working in gemini [Sun Sparc machine running Solaris], can you identify your friend in the dorm using a Pentium III running Linux?
2. **scalability:** If your beacon system becomes wildly popular (ala napster), can your system handle tens of millions of beacons?
3. **consistency:** How quickly do beacons realize when a beacon crashes so that the display that you get accurately reflects the beacons that are currently online?