CSCI 4900/6900 HW 3: Distributed Search

Assigned: Tuesday, Feb 6
Due: Tuesday, Feb 20, 11:00AM (LATE SUBMISSIONS NOT ACCEPTED)

1 Motivation

In the last two home work projects, we located other beacons that are currently online and accessible. We also implemented a simple beacon service that provides a file service. The next step is to be able to access files that are available in beacons that are not directly known to the current beacon. First, if you had used a central server based approach for Home Work 1, you will need to extend your beacons to be able to communicate with upto 2 different central servers. (If you had implemented your Home Work 1 using a peer-to-peer approach, then no change is required to add two servers).

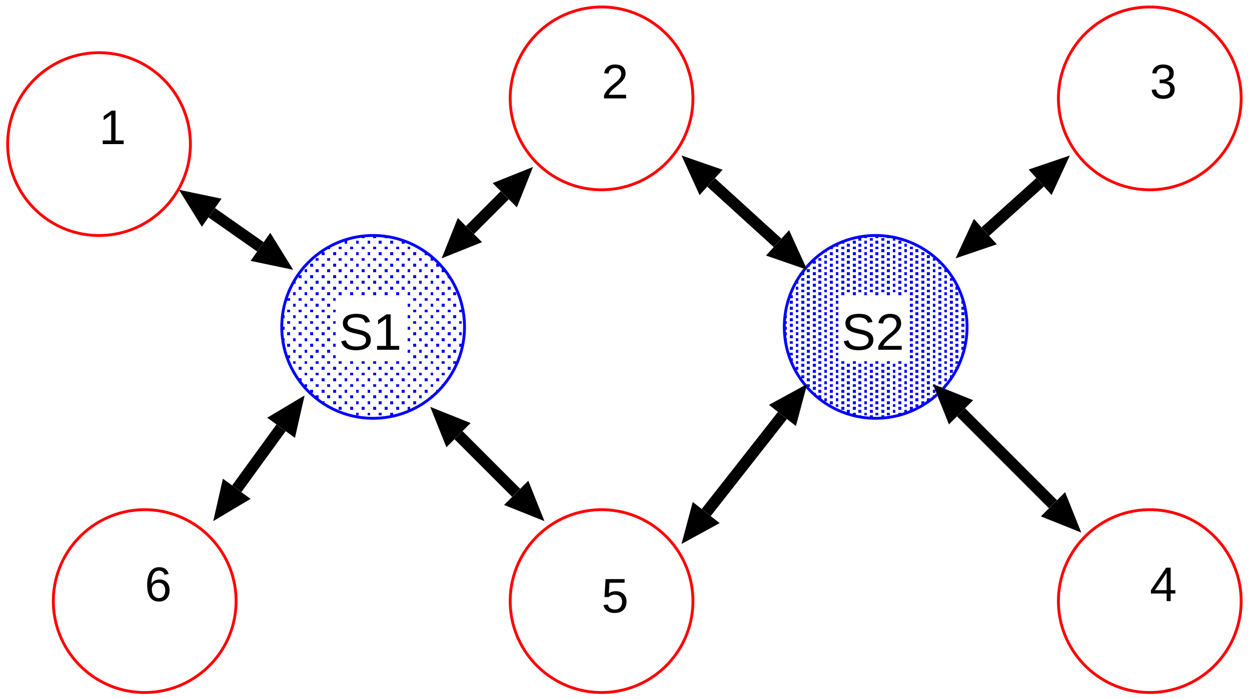

Figure 1: Beacons that can communicate with multiple beacond servers

A two server scenario is illustrated in Figure 1. In this scenario, beacons 1 and 6 communicate exclusively with *beacond* server S1, beacons 3 and 4 communicate exclusively with *beacond* server S2, while beacons 2 and 5 can communicate with both *beacond* servers S1 and S2. Using the modified central server based approach, beacon 1 knows that beacons 1, 2, 6 and 5 are online. However, beacon 2 knows that beacons 1, 6, 5, 3 and 4 are online by communicating with *beacond* servers S1 and S2. In this project, we will extend Home Work project 2 to let beacon 1 access files serviced by beacon 4 (even though beacon 1 does not directly know that beacon 4 is online). This extended serving mechanisms allows the beacons to search for services, even while the network is partitioned (as long as there is some path from the source to the destination).

2 Description

The goal of this project is to extend the beacon service to access files in beacons that are not directly accessible to them. For this project, the beacons will be extended to register with upto two different central servers (no change to add two servers is required for peer-to-peer implementations). We extend the services provided by Home Work 2 as follows:

- **searchget(token, serviceFile, hopCount)** If the requested file *serviceFile* is available in the beacon, the contents of the file are sent back. If the file *serviceFile* is not available, a recursive **searchget** is invoked by this beacon (on behalf of the requestor) on all the beacons that it knows of. Every such forwarding decrements the hopCount. Once the hopCount reaches 0 without successfully finding the file, the system returns an error message.

For example, in the scenario illustrated in Figure 1, suppose beacon 1 is searching for file hw1.tar available in beacon 4 (say). Beacon 1 will issue a searchget(token, 'hw1.tar', 5) to beacons 2, 5 and 6. Each one of them in turn, request hw1.tar from their neighbors till the file is located in beacon 4. Note that your implementation might return multiple copies of the same file. It might also return an FileNotFound error, even though there is a path available from the source to the destination. For example, suppose beacon 1 requests searchget(token, 'hw1.tar', 3), the request might get forwarded to beacons 2, 5 and 6 to return an FileNotFound error message (hopCount becomes 0), even though a route through beacons 2, 4 is feasible.

As usual, you are free to choose the exact technique to provide the service described above. You could use a traditional RPC style implementation or a multi-way RPC implementation (as discussed in the Active Names paper).

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 hw3 <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 hw3`. **Remember, I will randomly choose students who will be asked to explain their approach in person**. Issues that you might consider while developing and evaluating your system are:

1. **robustness:** How reliable is your system against failures? Does your beacon recognize forwarding loops? (wherein the requests are forwarded around in a loop without making progress towards the destination)
2. **scalability:** If your beacon 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, beacons repeatedly open connections to you to prevent you from servicing other, legitimate users)