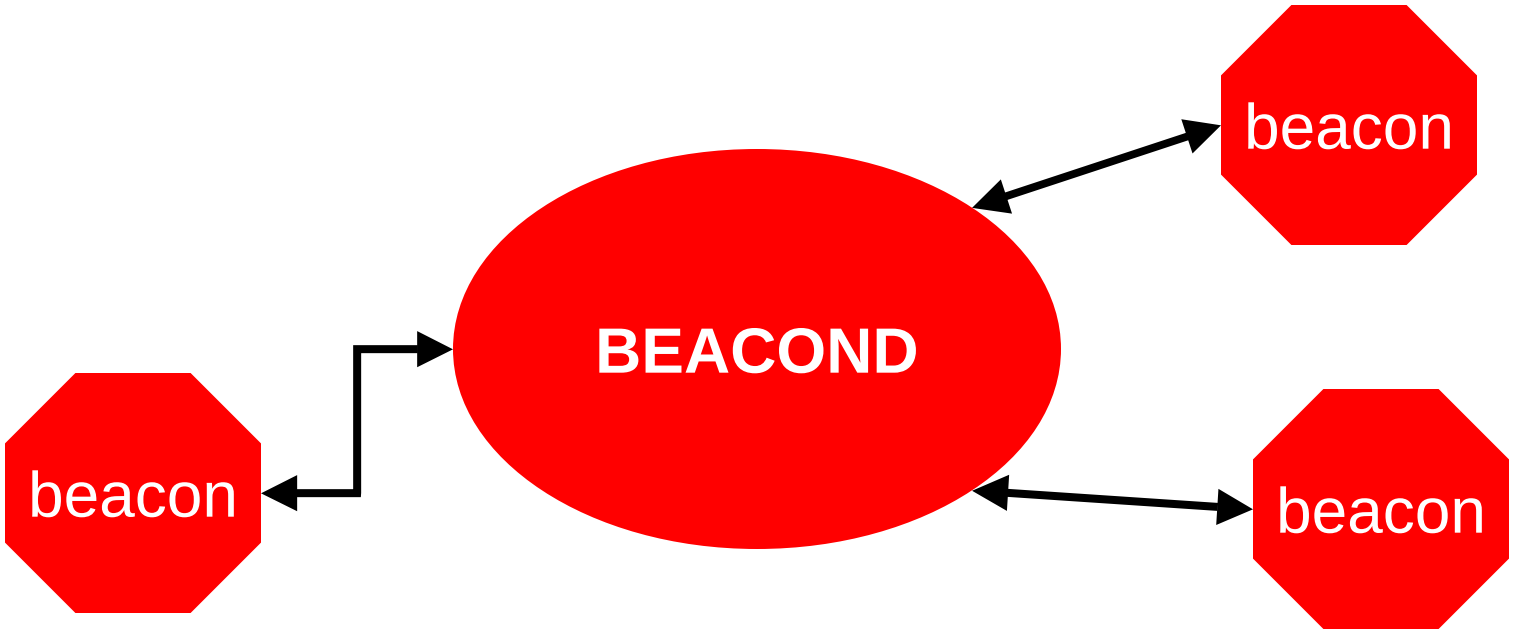
Announcements

- Homework #2 is due in 2 weeks. If your Homework 1 is not working (and you believe that it cannot be fixed); I will provide you with my implementation of Homework 1. You are free to use your own Homework #1 or start with my implementation.

Sample logic for HW #1



- Simple central server based approach

Beacond

Every beacon will register with the central beacond server.

- Every client will register with the server every δ time units.
- If a server never hears from a beacon for 3* δ time units, it assumes that the client has gone away.
- Once a client registers, the server will send back information about all the beacons that it knows about.

Beacond (pseudo code)

1. Open socket (soc1), bind to a known port and listen
2. while (forever)
3. Wait forever (accept on soc1) for read data on socket (soc2)
4. Garbage collect
    1. For all beacons
    2. if (currentTime – LastUpdateFromBeacon) > 3 δ delete information about this beacon
5. accept identification_t structure from beacon (soc2)
6. store the current time and this identification_t struct
7. for all identification_t structures
    1. Send (write) identification_t structure to soc2
8. close(soc2)

Beacon pseudo code

1. Open socket (soc1), bind to a random port, listen
2. Fill name, location and port (for soc1) in my identification_t
3. While (forever)
4. Open another socket (soc2), and connect to the hostname:port of the beacond server
5. Write (my identification_t structure) to soc2
6. While (! EOF)
7. read identification_t structure from soc2
8. if identification_t structure is new, PRINT NEW
9. if existing identification_t not seen, PRINT LEFT
10. Close(soc2)
11. sleep δ

-ilities

- Scalability: The central server is a performance bottle neck
- Consistency: upto 3 δ time units
- Interoperability: You have to be careful about sending integers in identification_t structure. Different machines (Sparc, Pentium) store integers differently. Utility functions like htonl(host to network - long) can help us interoperate with the identification_t structure

Homework #2

- In Homework #1, we developed a beacon system that “finds” other beacons that are currently online.
- The next important task for ubiquitous access to data is for the located beacons to do something useful on our behalf.
- In this homework, your beacons will provide a file service. It will implement five basic services: **open**, **list**, **get**, **put** and **close** to service the files.
- You should use the host:port in the identification_t structure (from Homework #1) to contact other beacons

Service syntax

- OPEN <key>

  To initiate any service, we have to first present our credentials as a <key> to the beacon. For our project, your beacon will accept any key. You beacon will return a token to signify a validated access. Further accesses to the beacon have to present this token for service.

- LIST <token>

  Lists all the files provided by this beacon

- GET <token>, <file>

  Retrieve a file that was listed using LIST command

- PUT <token>, <file>

  Uploads a file to the beacon. Any file that was PUT can be listed later using the LIST command

- CLOSE <token>

  Closes a session signified by the <token>

Home work #2

- In fact, after Home work #2, your beacons provide a rudimentary service similar to napster or gnutella (depending on whether you used a central server or a peer-to-peer approach in Home work #1)

Outline for today

- **Naming and Location Management**

1. **The Anatomy of a Context-Aware Application –** Andy Harter, Andy Hopper, Pete Steggles, Andy Ward and Paul Webster. AT&T Labs, Cambridge, UK

(we have seen the video about this Sentient project earlier in the course)

History of AT&T Cambridge Lab

- It used to be Olivetti Research Lab

- ORL developed active badges, VNC etc..
- VNC allows you to teleport across machines by moving the display to another terminal near you

Motivation for context-aware application

- Users application should be available where-ever the user goes, in a suitably adapted form
- Context aware application is one which adapts its behavior to a changing environment

  E.g. Follow-Me applications
- Context aware applications need to know the location of users and equipment, and the capabilities of the equipment and networking infrastructure

Components of Context-Aware Applications

- A fine-grained location system, which is used to locate and identify objects
- A detailed data model, which describes the essential real-world entities that are involved in mobile applications
- A persistent distributed object system, which presents that data model in a form accessible to applications

Components of Context Aware Applications

- Resource monitors, which run on networked equipment and communicate status information in a centralized repository
- A spatial monitoring, which enables event-based location-aware applications

Important lesson: It is very important to present location information in a form suitable to the application

Location system

- Out-door: Global Positioning Satellite (GPS)
  - Developed by US military, available for civilian use
  - Locates using triangulation with multiple satellites
  - Accurate up to a few meters. (Military version more accurate)

- In-door: Radio-based, Infra-red, Ultra-sound
  - For indoor systems, you have to worry about interference from multipath sources, obstructions and devices (such as fluorescent bulb ballasts)
  - Ultrasound is more immune to these interferences

Bat Unit

- Radio transceiver, ultrasonic transducer and control logic
- Each bat has a GUID
- Use the radio, ultrasonic transducer and the speed of sound in air (estimated from ambient temperature) to estimate location
- Use multiple receivers to get 3D location using multilateration
- Reflections of ultrasonic waves – statistical outlier elimination