Beacond: A Peer-to-Peer System to Teach Ubiquitous Computing

Surendar Chandra
Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN 46556
surendar@cse.nd.edu

Abstract

This paper describes a peer-to-peer (p2p) system (beacond) that is suitable for teaching important concepts in ubiquitous computing. The system exposes issues in peer location, p2p services, security and privacy issues. The system provided enough background to compliment class lectures and assisted students in designing their own course projects. Students continue to explore ideas exposed by beacond; some of these ideas are being further developed for publication in research conferences [1]. We present our experiences in using this framework for three different course offerings.

Categories & Subject Descriptors

K.3.2 [*Computer and Information Science Education*]: Computer science education

General Terms

Design, Experimentation

Keywords:

Ubiquitous Computing, Computer Networks

1 Introduction

Mobile devices (e.g. laptops, PDAs) using wireless network technologies offer a stepping stone to build ubiquitous and pervasive computing applications. Ubiquitous computing is a popular course being developed in several institutions to introduce the systems research issues in a *connected* future. This new course offering is designed to give students hands-on training in building and evaluating emerging systems through course projects.

The ubiquitous computing field is still in its nascent stages; course materials are typically designed around topical research papers. This course is built on the student's understanding of core operating systems and networking background. Even though this course is usually taught at an advanced graduate level, we adopted this topic for a senior undergraduate level class. The challenge in designing this course was to provide an hands-on introduction to our undergraduates who may not have the necessary background of a more advanced graduate class. A significant portion of our student population were active users of applications such as napster and gnutella. Hence, we wanted to leverage their familiarity with these applications to provide a hands-on experience with the ubiquitous technologies. Some of the fundamental ubiquitous computing technologies that we wanted to introduce to the students include:

- **Location management:** Locating other devices is usually the first step in accessing service provided by other devices. System entities need to be named first before being able to locate them. For example, in the cooltown project [2] beacons are identified using web urls. Cooltown utilized techniques such as direct and indirect sensing to announce the web presence of beacons.

- **Remote services in a partitioned p2p setup:** The ubiquitous devices provide a wide range of services to other devices that are in the vicinity.Typical ubiquitous devices utilize technologies such as IrDA (www.irda.org), Bluetooth (www.bluetooth.org) and offer limited range connectivity options to contact and receive services from nearby neighbors. For example, a printer might print a document from a PDA using a IrDA link. The range of these devices restrict the devices that can directly talk to each other. However, requests can be routed through other connected devices.

- **Security:** Security and privacy issues play an important role in allowing these connected devices to provide services to their connected peers. Unauthorized accesses are

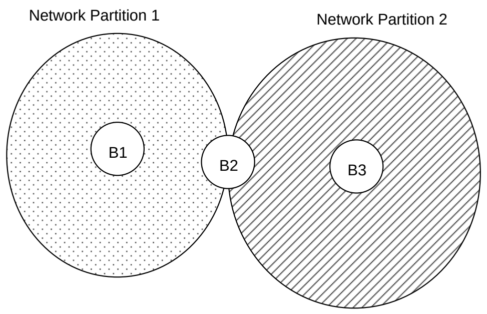

Figure 1: Beacond Architecture

especially an important problem for mobile devices that communicate with unknown neighbors.

Our goal was to develop a simple project that, not only compliments the class lectures for new technologies but also allow the students to further explore the techniques discussed in class. We wanted the system to provide basic infrastructure so that students can build further interesting services and applications. Our experience suggests that the system in fact, allowed students to further develop several application scenarios based on the technologies developed in this projects.

The remainder of the paper is organized as follows: we describe the general architecture of our beacond architecture in Section 2. Section 3 describes four project ideas and our experience in using them with a senior level undergraduate class. Section 4 outlines some ideas on how this architecture can be exploited to teach related courses such as computer networks and distributed systems. We present our conclusions in Section 5.

2 System Architecture

The beacond system was designed to be general enough to expose students to issues in location management, remote service discovery and security; while still amenable to students with limited systems exposure. The beacond architecture was influenced by research prototypes such as active badges [5], bayou [4] and cooltown [2].

The general system architecture is illustrated in Figure 1. The system consists of a number of beacons. Beacons are software representations of physical entities. For example, beacons can represent LCD projectors, people, microwave ovens, etc. Depending on the underlying network topology, not all beacons can communicate with every beacons. For example, in Figure 1 we note that, beacon B1 does not know about beacon B3 directly. Beacons B2 knows about all the other beacons directly because it belongs to both the network partitions 1 and 2. Each beacon provides certain services. For example, a printer beacon can print documents; an mp3 player can consume as well as provide mp3 files for sharing. Since beacons do not know about all other beacons; location and service queries have to be routed through other beacons. Care has to be taken in reducing the effects of such query flooding. Beacons should also make sure that service is only provided to the appropriate peer for auditing and access control purposes.

3 Our experience

We have used our infrastructure for 3 different course offerings, twice for “Ubiquitous Computing” and once for “Computer Networks”. For our class, we assigned four small individual projects based on the general system architecture; each of two weeks in duration. Each project built on techniques developed in earlier projects; with reference implementations for the past projects provided. Students were free to choose a programming language of their choosing; students chose Java, C, C++ and Perl running on Linux, Windows and Solaris platforms for their implementations. Students used the knowledge gleaned from these projects to design their own group course project. For all the projects, the students chose their own implementation strategy. The students were required to provide a written report explaining their particular approach and the interoperability and scalability tradeoffs.

In this section, we describe how the general beacond architecture described in the previous section was applied for our class projects. We present our experiences from student feedback. We also report our experience with the influence these projects had on students’ final course project. Next we will describe the four projects assigned in our class:

3.1 Locating beacons

The goal of this project was to locate beacons that were currently online and accessible. Beacons identify each other by exchanging their name, the Internet address and port number where the beacon can be reached. In general, there are a number of different ways for beacons to identify other beacons. Some popular techniques are:

1. **Central Server:** This approach utilizes a centralized server that keeps track of beacons that are online. Every beacon registers itself with this server. One would query this central server to search and locate other beacons. Beacons have to first know where this central server is located before they can locate other beacons (boot-strap problem). Beacons can keep this centralized location information updated using periodic updates initiated by the beacons or periodic polling queries from the central server.

2. **Peer-to-peer:** Here the beacons directly locate other beacons without any centralized data structures. Beacons can utilize multicasting (all beacons listen on the same multicast channel) 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.

The students were required to provide a written report explaining their particular approach to locate other beacons; how they solved the boot strap problem in a centralized approach or how they identified beacons outside the multicast range. They were also required to discuss the implications of their approach on *interoperability* (how the beacons recognizes beacons in a different host/operating system?), *scalability* (in case their beacon system becomes wildly popular (ala napster), can the system handle tens of millions of beacons?) and *consistency* (how quickly do beacons realize when a beacon crashes so that the results accurately reflect the beacons that are currently online?)

Discussion This project allowed students to catch up on their networking fundamentals; the project was conceptually simple enough that most students were able to utilize sample networking code from popular textbooks to build their system. Most students preferred the centralized approach; some adventurous students chose complex hierarchical multicast-based approaches. In order to level the playing field, we forced the students to uniformly use the p2p approach during the second iteration of the course offering, while increasing the assigned time. The students did not report significantly more difficulty with this p2p approach.

3.2 Accessing Remote Services

This project built on the earlier project (Section 3.1, locate other beacons that are currently online and accessible). Beacons knew where the other beacons were and how to communicate with them (the Internet address and port fields were exchanged as part of the location process). The goal of this project was for the beacons to provide a simple service. 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 etc), accept jobs for printing and provide status information (printer jam, out of paper, etc.). A more complex beacon can make requests to other beacons to accomplish its tasks. For example, the printer beacon can contact a clearinghouse to charge the user for the printing costs.

For our project, the beacons provided a service that will list and serve files on the Internet address and port exchanged as part of the location exchange. The beacons provided the following services:

- **open(key):** Beacons need a mechanism to authenticate and create an association with a particular beacon. Hence, all requests to a beacon were preceded by the *open()* function. For this project, we assumed that any key is valid. The beacons responded with a *token* that identified a particular session. Further requests to this beacon should be accompanied with this *token* for service. Requests without a valid *token* are denied.
- **list(token)**, **get(token, file)**, **put(token, file):** These services list all the files that were available at the beacon, download the contents of the requested file as well as upload the contents of the file, respectively. The students were free to choose the specific files serviced. Care must be taken to make sure that the files serviced with a *get()* request was among the files listed in the *list()* service. Requests for a file that is not available should be denied.
- **close(token)** This service will end the session with this particular beacon.

In fact, the beacons provide a rudimentary service for file exchange; similar to the services provided by systems such as gnutella, napster etc. As usual, the students reported their own implementation strategy and the implications of their approach on: *robustness:* (how reliable was the system against failures? If a beacon crashes and came back online immediately, can the system continue to provide service to existing clients?) *scalability:* (if the beacon suddenly becomes popular because of the files services, how much load can the system tolerate before the system crashes? Does the service degrade gracefully? Is the system immune to denial-of-service attacks?) and *security:* (how secure is the system regarding key management? If a beacon opens a session with another beacon and passed the session token to a friend, would they be able to utilize the key to access files? Can a beacon access files not listed by the *list()* service?)

Discussion This project built on students' experience in building network applications from the previous project. Many students preferred a centralized approach as it was deemed easier to implement and debug. Students frequently ran into buffer overflow and other common errors that allowed service to unlisted files. Similarity of this project to popular p2p applications such as gnutella and napster piqued students interest enough to develop fancy GUI extensions to improve the ease of using the system. This project also assisted the instructor in explaining mobile replications systems such as Bayou [4] and Coda [3].

3.3 Services in a partitioned network

The first two projects located other beacons that were online as well as provided a simple file service. The next step was the ability to access files that were available in beacons that were not directly known to the current beacon. Such agent applications can be built on top of a centralized or distributed implementations to locate other beacons. Beacons utilizing p2p approaches did not require any special modifications to locate beacons in a partitioned network. However, beacons using a central server based approach for locating other beacons were extended to be able to communicate with at least two different central servers.

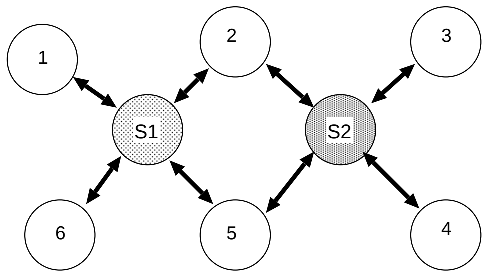

Figure 2: Beacons that can communicate with multiple centralized beacon location servers

An extended central server based approach utilizing two central servers is illustrated in Figure 2. In this scenario, beacons 1 and 6 communicate exclusively with *beacon location* server S1, beacons 3 and 4 communicate exclusively with *beacon location* server S2, while beacons 2 and 5 can communicate with both *beacon location* 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 *beacon location* servers S1 and S2.

In this project, we extended the beacon service mechanism to allow beacons to request service from beacons that are not directly accessible. For example, beacon 1 can 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).

The goal of this project was to extend the beacon service to access files in beacons that were not directly accessible to them. The services provided by beacons was extended as follows:

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

The students chose their particular search strategy. For example, in the scenario illustrated in Figure 2, suppose beacon 1 is searching for file hw1.tar available in beacon 4. Beacon 1 can 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 the particular 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 requested *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 and 4 is feasible and could have returned the requested file.

As usual, the students were free to choose their own implementation strategy. The students were required to provide a written report explaining their particular implementation strategy with regard to *robustness* (how reliable was the system against failures? Does the system recognize forwarding loops? (wherein the requests are forwarded around in a loop without making progress towards the destination)) and *scalability* (does the service degrade gracefully? Was it immune from denial-of-service attacks?)

Discussion This project was significantly more complex than the earlier projects. Students had more trouble successfully completing the project. However, the project sparked plenty of questions regarding the implications of such p2p searches on performance and security during the corresponding class lectures. The project formed a nice backdrop to lectures on distributed authentication and authorization schemes.

3.4 Security

Throughout the first three projects, beacons provided a mechanism for identifying themselves using a *name:key* pair. When a beacon logged into the system, it provided its *name:key* pair. On successful validation of this authentication key, a beacon is provided with an authorization token. This token was used for all further transactions.

In this project, the students provided an simple authorization service. They developed a key server that took the *name:key* pair to issue an authorization token. Each token was valid for a fixed time interval. The key server was also used to validate an authorization token. The beacons provided a new service that will utilize one or more key servers to validate authorization tokens. Such validation could allow beacons to selectively provide service to certain clients.

Key Service The key service provided simple authentication service. The key server provided authentication functionality to other beacons and hence need not provide an API for end user interactions. The students implemented the key server either as a single server or utilized p2p technologies to provide a robust key service mechanism. The key service provided the following functionality:

- **authenticate(name, key)** The key server authenticated the *name:key* pair and returned an authorization token. Each token was valid for a fixed time interval (set to 30 seconds for ease of testing).

- **validate(token)** The key server validated the authentication token and returned the *name* that was used to create this token. The key server should return an error message for invalid or expired tokens.
- **revoke(token)** The beacons can *revoke()* and invalidate tokens even before they expire. For example, a beacon can *revoke()* a token after the requested file was received, freeing up resources on the key service.

Beacon extensions The beacons were extended to utilize the authentication key service to implement the authentication primitives outlined in the first three projects (*get()*, *put()*, *close()* and *searchget()* services). Hence, services that used to work might fail because of an expired or an invalid token. For this project, the beacons were also extended to provide the following service for ease of testing:

- **validate(token)** The beacons utilized the key service to *validate()* a token. The beacon printed the name associated with the token or an error message for invalid or expired tokens.

As usual, the students were required to provide a written report explaining their particular approach. They were also required to discuss the implications of their approach on robustness and scalability issues.

Discussion This project benefited from corresponding class lectures on authentication and authorization issues for ubiquitous information access. In fact, based on the prior experience with the first three projects, several students actually suggested the general idea for this project. Prior to this project, the consistent complaint from students was the apparent lack of detailed security in typical ubiquitous computing prototypes. This project helped drive the ideas of trusted computing bases and the need for end-to-end authorization through companion lectures. Students learnt to appreciate the fact that end-to-end security is much more complex than simple authentication.

3.5 Overall experience

In general, we found these projects better complemented the class lectures. For their own course projects, the students further developed several application scenarios that exploited the beacond system for realistic applications. The parallels between beacond and popular p2p systems such as gnutella, napster etc. piqued students interest in further understanding those technologies. Some of the students are exploring research issues identified by the beacond infrastructure.

4 Future Educational Use

We believe that the technologies exposed by beacond system can be utilized in the instruction of courses such as computer networks and distributed systems. The p2p nature of beacond lends itself to teaching mobile and ad-hoc scenarios in computer networks. For our course, the project benefited from similarity to popular p2p applications. Such interest led the students to go beyond the stated project requirements to think of better ways to implement, say napster, while achieving the instructors teaching goals for this course. We believe that this enthusiasm would translate to similar positive results for related systems courses.

5 Conclusions

In this paper, we describe a distributed p2p system that was utilized to compliment class lectures in topics in ubiquitous computing. The system provided hands on experience to important system technologies. The system allowed students to further develop several application scenarios based on the technologies developed in this projects. We also feel that this system has further applicability in teaching related systems courses such as computer networks and distributed systems. The reference code and project descriptions will be available online at the instructors' course web page.

Acknowledgments

We wish to thank the students for their valuable comments and feedback. Beacond was developed while the author was a faculty at the University of Georgia, Athens, GA. We would also like to thank Norma Brown for her helpful insights.

References

[1] Collins, E., and Chandra, S. Ringnet: A generalized scheme to maintain path-connected topology in dynamic p2p networks. (in preparation), 2002.

[2] Kindberg, T., Barton, J., Morgan, J., Becker, G., Caswell, D., Debaty, P., Gopal, G., Frid, M., Krishnan, V., Morris, H., , Schettino, J., Serra, B., and Spasojevic, M. People, places, things: Web presence for the real world. In *Workshop on Mobile Computing Systems and Applications (WMCSA)* (Monterey, CA, Dec. 2000).

[3] Satyanarayanan, M., Kistler, J. J., Kumar, P., Okasaki, M. E., Siegel, E. H., and Steere, D. C. Coda: A highly available file system for a distributed workstation environment. *IEEE Transactions on Computers 39*, 4 (Apr. 1990).

[4] Terry, D. B., Theimer, M. M., Petersen, K., Demers, A. J., Spreitzer, M. J., and Hauser, C. H. Managing update conflicts in a weakly connected replicated storage system. In *Proceedings of the 15th SOSP* (Dec. 1995).

[5] Want, R., Hopper, A., Falcao, V., and Gibbons, J. The active badge location system. *ACM Transactions on Information Systems 10*, 1 (Jan. 1992), 91–102.