

BonAHA: Service Discovery Framework for Mobile Ad-Hoc Applications

Suman Srinivasan, Arezu Moghadam and Henning Schulzrinne,
Columbia University, New York, NY
{sumans,arezu,hgs}@cs.columbia.edu

Abstract—In today’s mobile networks, devices often move from one network to the next, forming transitory associations without a fixed support infrastructure. The state in this network frequently changes due to node transitions, and nodes are always meeting new peers. In this scenario, traditional models for writing networking software, such as the client-server model or even the peer-to-peer model, turn out to be unsuitable for writing such “ad-hoc applications”. A new framework needs to be developed for this class of applications to be aware of node transitions as well as metadata (properties or key-value pairs) that the nodes possess. In this paper, we present a framework we have built for this class of applications. Our library, called BonAHA (Bonjour for Ad-Hoc Applications) aims to be easy and intuitive, abstract the ad-hoc networking details and at the same time provide flexibility to the developer to develop powerful and rich ad-hoc applications.

I. INTRODUCTION

For software applications to work properly in highly transitory networks, it is necessary to maintain some sort of awareness of network state and discover nodes entering and leaving the network. This requires multicast queries with multicast/unicast responses in order to keep track of nodes in the network.

Service discovery protocols provide a simple framework to match our requirements. However, raw service discovery APIs, while suitable for writing applications that announce and browse for services, are not inherently suitable for writing ad-hoc applications. This is because service discovery protocols only address service announcement and discovery, and require the developer to implement the details of monitoring network transitions, which could quickly become very tedious.

With the BonAHA framework we have developed, we provide a framework for easy development and deployment of such ad-hoc applications that work in transitory networks. BonAHA is built on top of a popular set of service discovery protocols which use multicast DNS (mDNS) [1], the most popular implementation of which is a library by Apple Computer called Bonjour [2].

In this paper, we will present our motivations for developing the BonAHA API in Section II. In Section III, we introduce service discovery, its features and its shortcomings when applied to the highly mobile network. We focus on mDNS.

In Section IV, we introduce our BonAHA framework, compare its API to that of service discovery and show how BonAHA is much simpler and intuitive for developing ad-hoc applications. In Section V, we discuss sample applications that we have written using the BonAHA API and present sample

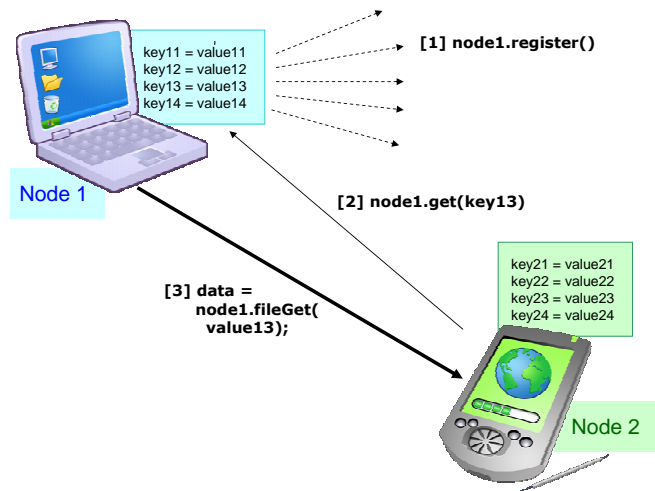


Fig. 1. The overall architecture of BonAHA from the developer’s perspective. The API allows the developer to treat the network as a set of objects with associated metadata. Network events such as nodes entering and leaving the network can be handled through simple event handling functions. The diagram shows how events are announced and browsed (1), how metadata is set and obtained (2), and how network communication is performed based on the above operations (3).

code to show how it can be used. Section VI discusses future work on BonAHA.

II. MOTIVATION

An overview of the BonAHA framework we have developed is shown in Figure 1. BonAHA exposes the nodes in the network as objects that can be accessed using object-oriented function calls. It also enables the developer to handle network events such as nodes entering and leaving the network, as described later in the paper.

In this paper, we are mostly concerned with the problem of applications running in what we call “opportunistic networks”. We define “opportunistic networks” as networks in which nodes remain disconnected at times, or intermittently connected for short periods of time. The networks thus have a characteristic of being highly transient with nodes moving in and out of the local area network or subnet. As we can see from the widespread deployment and popularity of mobile devices and location-based services, opportunistic networks are already becoming commonplace.

For networked applications to function consistently in opportunistic networks, they need to be aware of changes in the

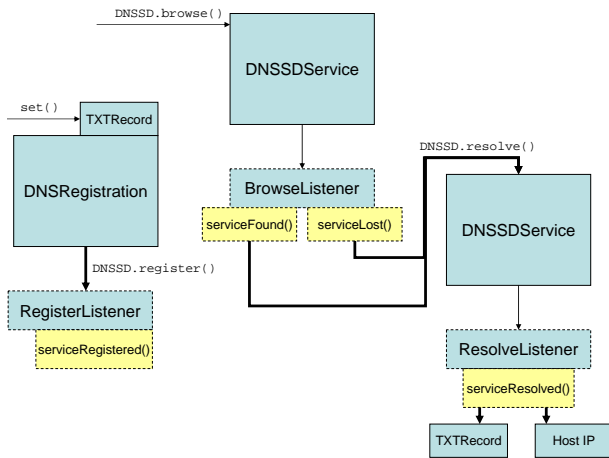


Fig. 2. The state diagram for the Bonjour API.

network, such as nodes entering or leaving the network, and changes in key-value pairs associated with that node.

In order to develop applications that support the intermittently connected network scenario, an application developer will have to write code that implements and handles several networking events. In fact, when we implemented our first version of an application to allow distributed web access and e-mail delivery [3], we coded multicast packets for announcement and discovery manually.

However, it becomes tedious to rewrite the same code and maintain all internal network state when developing ad-hoc applications that run in opportunistic networks. It was obvious from our initial development that a software library is needed so that development of ad-hoc applications becomes easier.

We found that the service discovery most suitable for our purpose is an implementation of multicast DNS in the form of Bonjour, an open-source technology from Apple Computer, which is implemented and runs on Mac OS, Windows, Linux, Unix variants and several other platforms.

However, we found that the mDNS protocol still has some shortcomings for developing ad-hoc applications. The developer still has to deal with details of service discovery that require a learning curve and maintaining unnecessary data structures in code.

Based on our use of mDNS for building mobile ad-hoc applications [3], we find that implementing a truly mobile application is difficult to develop unless the developer completely understands how mDNS works.

Our BonAHA framework runs on top of mDNS and is suitable for writing ad-hoc applications. By handling the details of the service discovery protocol, BonAHA allows the developer to focus on developing ad-hoc applications by allowing him or her to easily keep track of network state, the nodes in the network, as well as the metadata associated with each node.

III. SERVICE DISCOVERY

Service discovery refers to protocols which enable automatic detection of devices and services on a computer network.

Service discovery is fairly mature technology and has been around for over a decade. Service discovery protocols range from lightweight protocols such as DHCP [5] to heavyweight protocols like JXTA [6].

However, service discovery protocols that can be applied to writing ad-hoc applications that run in opportunistic networks are very few in number. Most protocols are too heavyweight for use in a limited-device, intermittent network scenario. Further, most of these protocols require some sort of bootstrap node or protocol which is not practical in intermittent and especially highly mobile network scenarios. Hence they are not suitable for use in ad-hoc applications that run in opportunistic networks.

Among the limited set of service discovery protocols that are suitable for ad-hoc applications, the Bonjour implementation seems to be the most mature and stable protocol set. Further, Bonjour has been implemented on most major platforms, including Windows, Mac, Linux and POSIX platforms.

A. mDNS and Bonjour API

The Bonjour API, while simple for developing mDNS applications, requires a learning curve and has some shortcomings that make it not completely applicable to opportunistic networks.

The fundamental reason why Bonjour, in its native form, is not suitable for ad-hoc applications is that it is primarily concerned with changes in the network, such as services entering and leaving the network. It makes no attempt at capturing and maintaining a view of the network state, which is important for applications working in opportunistic networks to be developed easily.

The following is a list of shortcomings in the Bonjour API which require some more work (this is in reference to its Java API):

- Bonjour has three interfaces and five event-handling functions, all of which have to be completely implemented by an application developer who is attempting to implement an ad-hoc application.
- Bonjour makes no attempt to capture and store the state of the network. Even though the Bonjour API enables the developer to handle network events, it does not internally store this state of the network, such as nodes present and associated metadata. The developer has to maintain this state himself.
- Bonjour requires a two-step process for resolving a node entry or exit. In order to access node details and metadata, node entry needs to be chained with another function that resolves the node details.

IV. BONAHA

In order to enable developers to easily develop mobile ad-hoc applications, a framework that encapsulates Bonjour and presents an easy API for developers is needed.

We are working on building a comprehensive framework called BonAHA to address this, and have completed and released the first version that addresses the needs outlined

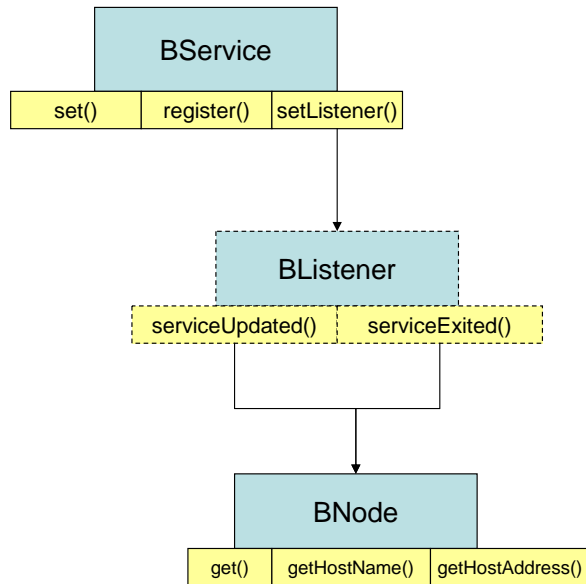


Fig. 3. The state diagram for the BonAHA API.

Listing 1 The details of the BonAHA API.

Classes:

BService: Description of a DNS-SD service

BNode: A node that offers a *BService*. Note that while a *BNode* maps to a physical node, there may be a many-to-one relationship as each physical node can offer multiple services, and hence correspond to multiple *BNodes*.

Interfaces:

BListener: Handling network events such as services entering or exiting the network

Callback functions for events

BListener

serviceUpdated() when services appear or are updated

serviceExited() when services leave

earlier in the paper. We have released BonAHA under the GPL license on Sourceforge [7].

As part of our earlier work, we completed a GUI library called BonSwing [8] which provides developers a GUI library to build applications for opportunistic networks. The library and sample applications are also available for download from SourceForge [9].

A. Architecture

BonAHA uses a simple concept of "service," which is very similar to the service defined for the mDNS protocol. Applications can register or listen to a particular service on a network. This service is instantiated as a string name. The name is the same as the DNS-like names used in DNS-SD's service names. As an example, a node announcing a HTTP server service would have a DNS service name `_http._tcp`.

Metadata for the node is set using suitable object-oriented function calls. Services on the network can be discovered by instantiating a service object and registering it to respond to network events, such as a node update or nodes arriving or exiting the network. Metadata for neighboring nodes in the network can be obtained by using issuing object-oriented function calls.

The API used for handling network events in BonAHA is given in Listing 1. The state diagram is outlined in Figure 3.

An application that is announcing a service will usually follow the following steps:

- Create a service object with the name of the service
- Set any metadata associated with that service
- Register it

An application that is listening to service announcements on the network will usually follow the following steps:

- Create a service object with the name of the service
- Set an event handler object for this service
- The class handling events for the service will handle events corresponding to node updates (which includes nodes arriving in a network) and node exits
- Metadata associated with that node can be retrieved from the nodes

Using the BonAHA library, an application developer would be able to completely encapsulate network node and metadata event handling in a few lines of code. Without this framework, he would need to understand Bonjour or other service discovery protocols thoroughly and implement at least several dozen lines of code to listen to network events and node arrival and departure.

B. BonAHA API

The BonAHA API aims to present the network to the developer as a set of objects with metadata which enter and leave the network, thus triggering events. An outline of the API can be seen in Figure 3 and in Listing 1.

The *BService* class allows one to construct a service instance which corresponds to a DNS-SD service in the network. Its constructor allows for easy creation of such service instances without needing to know the full DNS name of the service.

The *BListener* interface handles two types of events in the network: node entry (or update) and node exit. Node arrivals are treated the same as node updates, since this simplifies the event handling without sacrificing any functionality. The two functions that are handled by this event are *BListener.serviceUpdated()* and *BListener.serviceExited()*, both of which return a *BNode* object.

The *BNode* object corresponds to a node in the network offering the service requested, and exposes the metadata (key-value pairs) of the node, such as its host name, host address, service name offered as well as the metadata (key-value pairs) of the node.

While the *BNode* maps to a physical node, it actually corresponds to a particular service type that is advertised by the physical node. For instance, a physical node would return

two *BNode* objects if it was offering two services of the same type. Similarly, a physical node would return a unique instance of a *BNode* for each service type it was offering.

A developer who needs to offer an instance of, or view instances of, a service type in the network, would create a *BService* object. After creating this object, he would do either or both of the following operations:

- Register a service with *BService.register()*
- Listen to network events for the service type with *BService.setListener*

The *BService.setListener*, which takes an object that implements a *BListener* interface, associates the network events with the implementation's function calls.

Program 1 The program for implementing location updates using the BonAHA API. Only the code using the BonAHA API for service announcement and updates is shown.

```
public LocationFinder() {
    // Create the BonAHA service and set metadata
    service = new BService('7ds_location', 'tcp');
    // Set my location
    service.set('Latitude', lat);
    service.set('Longitude', lon);
    service.register(); // Register myself
    // Listen for new nodes on the network
    service.setListener(this);
}

/* Another node enters or updates its location */
public void serviceUpdated(BNode n) {
    System.out.print('Updates from: '+n.getHost());
    // Get the node's location metadata
    String nodeLat = n.get('Latitude');
    String nodeLong = n.get('Longitude');
    // Process peer's information
}

/* When a node leaves the network */
public void serviceExited(BNode n) {
    System.out.println(n.getHost() + ' (' +
        n.getHostAddress() + ') ' + left the system');
}
```

Using the BonAHA API, we have developed ad-hoc software applications such as a automatic location finder, and a simple networked Tic-Tac-Toe game that uses only BonAHA to update game status.

V. SAMPLE APPLICATIONS

A. Location Finder

The LocationFinder application is a simple command line application. The presumed scenario for this sort of application is where a device does not have access to GPS data and only has access to the location information of nearby nodes. Using this information, it updates its own location by averaging the location of all the other nodes in the neighborhood.

We have written the Location Finder application using the Bonjour API as well, and for equivalent functionality, the Bonjour code required twice as many lines as code as the BonAHA code.

Some of the basic code for running the Location Finder application can be found in Program 1.

B. Tic Tac Toe

While multiplayer games like Tic-Tac-Toe are rather easy to write using sockets and other forms of network programming, our BonAHA framework exposes an entirely new way of writing such multiplayer games.

We will briefly explain how the multiplayer functionality of the game works. First, a *BService* object is created and *register()* is called to announce its availability on the network. A listener interface is also attached to listen to network events.

Upon receiving a *serviceUpdated()* event, the event handler code first processes which node the event update is from. Next, it processes the metadata from the incoming node and maps it to an internal data structure representing the location of the players' moves, which is used to update the game.

Some of the networking code used to create the networked Tic-Tac-Toe application can be found in Program 2. A screenshot of this program is shown in 4.

A traditional multiplayer Tic-Tac-Toe or other networked game would require the developer to write client and server sockets and process data packets. With the BonAHA framework, the developer is able to implement the networking functionality in four lines for the Tic-Tac-Toe game.

The BonAHA framework enables the developer to handle nodes entering and leaving the network. Our Tic-Tac-Toe game is able to automatically terminate a game when a user leaves, and wait for and detect when another user wants to join. This feature was added with just three lines of code.

Program 2 The code for getting the list of values from neighboring nodes for the Tic-Tac-Toe game.

```
/* When I make a move */
public void mouseReleased(MouseEvent e) {
    // Get the location where user wants to move
    int col = (e.getX() * 3) / getSize().width;
    int row = (e.getY() * 3) / getSize().height;

    // Update my node's metadata to reflect state.
    // This is announced on the network
    service.set(col+"",row, col+"",row);
}

/* New player; or other player made a move */
public void serviceUpdated(BNode n) {
    // Check which player made a move.
    // Then get node values (player positions)
    String[] values = n.getValues();

    // Update internal data structure

    // Repaint the game
    this.repaint();
}

/* Player has left */
public void serviceExited(BNode n) {
    // Display message; wait for new player
}
```

C. 7DS - File Synchronization

We are continuing to work on a class of applications called 7DS [3] that provide some useful applications in the absence of a connected network. One such application is a file

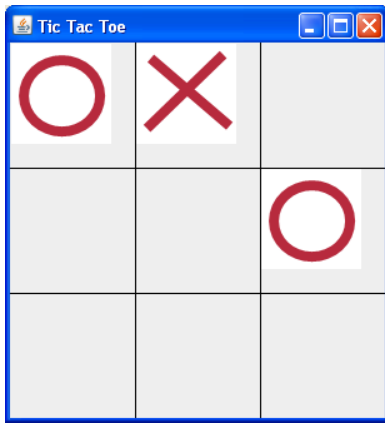


Fig. 4. A screenshot of the Tic-Tac-Toe game developed using BonAHA. While it looks similar to any regular networked two-player Tic-Tac-Toe game, it is much simpler to develop due to the BonAHA framework.

synchronization application [10]. This application allows users to synchronize files that are placed in a shared folder. When nodes come into each other's vicinity, they automatically synchronize the files so that each of them has the latest versions of the files. The code for this application is also available for download from the BonAHA Sourceforge website [7].

VI. FUTURE WORK

Our current version of the BonAHA library is written using Java on top of the Bonjour library. However, the concepts presented in this paper relate to a generic application framework, and we would like to see BonAHA extend to, and be used in, different environments on different programming languages.

For example, BonAHA can be easily extended to run on Avahi [4] for Linux natively, or on a Java implementation of mDNS called jmdns [11] ported to Google Android and J2ME-enabled mobile devices as well.

In addition, it may be possible to extend the BonAHA framework or API functions to run on top of completely lightweight hardware discovery protocols such as Bluetooth as well. Currently, the BonAHA library works on top of mDNS, which works on any network that supports the IP protocol. By porting the BonAHA library to run on different protocols (such as ZigBee, Bluetooth) and different programming languages, we can enable the ad-hoc application developer to truly learn one set of APIs and be able to run his program on any platform and hardware that supports BonAHA.

VII. RELATED WORK

JXTA [6] is a peer-to-peer framework implementation made by Sun Microsystems for the Java platform. JXTA is a very powerful framework and has been implemented in J2ME for mobile platforms. However, JXTA does not have the necessary framework to handle network events, such as nodes joining or leaving the network, which is very necessary for our class of applications. In addition, the JXTA protocols appear to be more suited to maintaining distributed metadata in always-connected networks and hence more suitable for heavyweight applications such as file-sharing programs.

Peer2Me [12] is a implementation of a mobile ad-hoc framework using JXTA. It appears to overcome the problem of network discovery using a platform-specific network library for detecting Bluetooth network changes. However, it is currently limited to J2ME devices and devices that use Bluetooth.

The only framework that is so far closely comparable to our work is the recently released LightPeers framework [13]. LightPeers is a library framework written in Mono, and runs on Windows Mobile, Windows and Mac platforms. LightPeers enables development of mobile peer-to-peer and ad-hoc applications. There are some minor differences, in that LightPeers' discovery mechanism uses an announcement packet every second to check for nearby devices, while our mDNS-based platform uses exponential backoff to reduce the number of packets over time. Further, the BonAHA API allows developers to get and set metadata associated with each node, which allows the developer to treat the entire network as a collection of objects and network events. LightPeers does not have such a feature.

VIII. CONCLUSION

We believe that the BonAHA framework provides a promising start for easing development of highly mobile ad-hoc applications, the need for which is emerging fast in today's highly mobile and connected, yet, transient world. We also hope that our continuing work on the BonAHA framework will enable complete ad-hoc applications to be built easily with minimal overhead for the application developer.

IX. ACKNOWLEDGMENT

This work was supported by the National Science Foundation (NSF) under Grant 04-54288.

REFERENCES

- [1] Zeroconf working group. [Online]. Available: <http://www.zeroconf.org/>
- [2] "Apple Computer's Bonjour." [Online]. Available: <http://developer.apple.com/networking/bonjour/>
- [3] S. Srinivasan, A. Moghadam, S. Hong, and H. G. Schulzrinne, "7DS - Node cooperation and information exchange in mostly disconnected networks," in *IEEE International Conference on Communications (ICC)*, 2007.
- [4] Avahi. [Online]. Available: <http://avahi.org/>
- [5] "Dynamic host configuration protocol." [Online]. Available: <http://tools.ietf.org/html/rfc2131>
- [6] "Project jxta." [Online]. Available: <http://jxta.org/>
- [7] BonAHA download page. [Online]. Available: <http://bonaha.sourceforge.net/>
- [8] S. Srinivasan and H. G. Schulzrinne, "BonSwing: A GUI Framework for Ad-Hoc Applications Using Service Discovery," in *ACM International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, 2007.
- [9] BonSwing download page. [Online]. Available: <http://bonswing.sourceforge.net/>
- [10] A. Moghadam, S. Srinivasan, and H. Schulzrinne, "7DS - A Modular Platform to Develop Mobile Disruption-tolerant Applications," in *IEEE NGMAST 2008*, Wales, UK, June 2008.
- [11] "Jmdns." [Online]. Available: <http://jmdns.sourceforge.net/>
- [12] A. I. Wang, T. Bjornsgard, and K. Saxlund, "Peer2me - rapid application framework for mobile peer-to-peer applications," in *International Symposium on Collaborative Technologies and Systems 2007*, 2007.
- [13] B. G. Christensen, "LightPeers: A Lightweight Mobile P2P Platform," in *Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications Workshops*, 2007.