DESIGN AND ANALYSIS OF APPLICATION ARCHITECTURE
FOR OPPORTUNISTIC NETWORKS
USING AD HOC Wi-Fi
by
SANKALP VINOD SHERE

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF SCIENCE IN COMPUTER ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON
December 2009
ACKNOWLEDGEMENTS

It’s a great pleasure to get to thank everybody who has directly or indirectly contributed to this thesis. First I would like express my gratitude towards my thesis supervisor, Dr. Yonghe Liu. I am truly thankful to him for his constant encouragement, guidance and patience. He always showed enthusiasm and took great efforts in helping me understand problems and come up with solutions.

I would also like to thank my student colleagues, Jing Wang and Na Li for creating positive work environment in CReWMaN lab and being available to discuss problems. I am grateful to my graduate advisors Mr. Mike O’Dell and Dr. Bahram Khalili for their helpful advice throughout MS degree program.

Lastly, and most importantly I would like to thank my friends, family and well wishers for always being there for me and believing in me. It is their continuous support and encouragement that has made this thesis possible. To them, I dedicate this thesis.

November 23, 2009
ABSTRACT

DESIGN AND ANALYSIS OF APPLICATION ARCHITECTURE
FOR OPPORTUNISTIC NETWORKS
USING AD HOC Wi-Fi

Sankalp Shere, M.S.
The University of Texas at Arlington, 2009

Supervising Professor: Dr. Yonghe Liu

In recent years, the number of smartphone users has increased by many folds. In fact, it is estimated to reach 100 million by 2013. Current generation of smartphones has better storage, battery, computing capabilities and they come equipped with short range communication technologies like Bluetooth and Wi-Fi. These improved capabilities coupled with staggering rise in number of users, has prompted growing interest in smartphone applications that help users to communicate with each other directly and over the internet. In particular as a result, when a large number of mobile users can communicate directly with each other during opportunistic contacts when they are physically proximate to each other, an opportunistic network is formed.

Consider scenarios where researchers are gathered at a research conference or shoppers are roaming in mall. It will benefit them significantly if they can automatically exchange information of their interest with neighboring users. For example, researchers can receive information about nearby researchers with similar research interests, and shoppers can get to know about discounts on products or brands they like. All these scenarios make perfect case for opportunistic networking applications.
In this thesis, we design and implement an application architecture that addresses the needs of the opportunistic networking applications. We provide innovative information publishing model that embeds application and device information in MAC layer frames and uses faster MAC layer device discovery mechanisms to gather application level information. Based on recent signal strength values and history of time of contact between nodes, we design different neighbor selection strategies and analyze their performance by conducting series of experiments in different environments. Finally we enhance an existing multicasting communication framework to implement connection establishment and one-hop message exchange in opportunistic networks.

Instead of relying on Bluetooth communication technology where many implementations exist for opportunistic networks, we focus on wireless LAN (Wi-Fi) technology owing to its longer communication range, hence longer opportunistic contact period, more contact opportunities and faster device discovery technique. As we will reveal, the difficulty of designing and implementing over Wi-Fi is significantly higher as compared those based on Bluetooth.

The first component of our architecture is neighbor discovery. In highly dynamic and ever changing opportunistic networks, efficient and timely neighbor discovery provides the foundation for any following data communications. In our design, we employ current signal strength values and history of time of contact information to facilitate neighbor discovery and selection. Using innovative beacon stuffing mechanism, we publish application and device information using MAC layer beacon frames, thus speeding up device discovery process.

The next component in our architecture addresses the need of communication among devices. We design connection establishment mechanism for a device to initiate connection with its best neighbor on unique Wi-Fi SSID network. Our architecture handles one-hop message exchange over UDP/IP protocol, also maintains communication message synchronicity. We keep track of all successful and unsuccessful communication sessions and
assign credibility to neighbors for future encounters. We realize the intermittent nature of opportunistic communication and provide re-establishment mechanism for the broken communication links between neighboring devices.

To demonstrate and verify the performance of our architecture, we have implemented the prisoner's dilemma game as an example application. In this game, a mobile node finds suitable devices in the surroundings and plays autonomous games with its neighbors. Also, we have developed a device discovery application that lets user configure neighbor discovery parameters.

We have conducted series of functional and performance tests to analyze device discovery scheme, different neighbor selection mechanisms and the efficiency of communication between neighboring devices. Our results show that using our neighbor discovery and selection mechanisms, speed and performance of the opportunistic communication, especially neighbor discovery process can be significantly improved. Using signal strength and time of contact information, our neighbor selection strategies give great insight into still unexplored aspect of opportunistic networks – neighbor selection.
# TABLE OF CONTENTS

ACKNOWLEDGEMENTS ........................................................................................................ iii

ABSTRACT ................................................................................................................................. iv

LIST OF ILLUSTRATIONS ......................................................................................................... ix

LIST OF TABLES ....................................................................................................................... xi

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>2. BACKGROUND</td>
<td>7</td>
</tr>
<tr>
<td>2.1 Opportunistic Networks</td>
<td>7</td>
</tr>
<tr>
<td>2.2 Wireless LAN Protocol</td>
<td>13</td>
</tr>
<tr>
<td>3. RELATED WORK</td>
<td>21</td>
</tr>
<tr>
<td>3.1 Beacon Stuffing</td>
<td>21</td>
</tr>
<tr>
<td>3.2 MultiCommFramework</td>
<td>25</td>
</tr>
<tr>
<td>3.3 iClouds</td>
<td>30</td>
</tr>
<tr>
<td>4. APPLICATION ARCHITECTURE</td>
<td>33</td>
</tr>
<tr>
<td>4.1 Problem Specification</td>
<td>33</td>
</tr>
<tr>
<td>4.2 Design Consideration</td>
<td>35</td>
</tr>
<tr>
<td>4.3 Design and Implementation</td>
<td>37</td>
</tr>
<tr>
<td>4.3.1 OppCommNetworking</td>
<td>38</td>
</tr>
<tr>
<td>4.3.2 OppCommFramework</td>
<td>46</td>
</tr>
<tr>
<td>4.3.3 Opportunistic Networking Application: Iterated Prisoner’s Dilemma</td>
<td>48</td>
</tr>
<tr>
<td>4.3.4 Development and Testing Environment</td>
<td>51</td>
</tr>
</tbody>
</table>
5. EXPERIMENTS AND RESULTS .................................................................................. 52

5.1 Opportunistic Networking Application: Device Discovery ................................. 52

5.2 Device Discovery Comparison ............................................................................. 53

5.3 Neighbor Selection Performance Evaluation ...................................................... 53

5.3.1 Experiment Setup ........................................................................................... 55

5.4 Credibility Tests ................................................................................................. 61

5.5 Network Statistics ............................................................................................. 63

6. CONCLUSION AND FUTURE WORK ................................................................. 64

REFERENCES ............................................................................................................ 66

BIOGRAPHICAL INFORMATION .............................................................................. 69
# LIST OF ILLUSTRATIONS

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Opportunistic Network Example</td>
<td>9</td>
</tr>
<tr>
<td>2.2 Typical IEEE 802.11 Architecture</td>
<td>14</td>
</tr>
<tr>
<td>2.3 IEEE 802.11 Protocol Architecture</td>
<td>17</td>
</tr>
<tr>
<td>2.4 Network of independent BSSs</td>
<td>20</td>
</tr>
<tr>
<td>3.1 Message fragment in beacon stuffing protocol</td>
<td>22</td>
</tr>
<tr>
<td>3.2 Important fields in IEEE 802.11 beacon packet</td>
<td>22</td>
</tr>
<tr>
<td>4.1 Bluetooth Vs Wi-Fi Comparison</td>
<td>36</td>
</tr>
<tr>
<td>4.2 Application Architecture</td>
<td>37</td>
</tr>
<tr>
<td>4.3 Stuffed SSID field of IEEE 802.11 beacon</td>
<td>39</td>
</tr>
<tr>
<td>4.4 NeighborRecord data structure</td>
<td>39</td>
</tr>
<tr>
<td>4.5 Flowchart for the node sending Session Initiation Request</td>
<td>42</td>
</tr>
<tr>
<td>4.6 Flowchart for the node replying to Session Initiation Request</td>
<td>43</td>
</tr>
<tr>
<td>4.7 Session Initiation Request Message Format</td>
<td>44</td>
</tr>
<tr>
<td>4.8 Session SSID - Stuffed SSID field of IEEE 802.11 beacon</td>
<td>45</td>
</tr>
<tr>
<td>4.9 RadioController – Wrapper to control Bluetooth and Wi-Fi radio operations</td>
<td>45</td>
</tr>
<tr>
<td>4.10 WiFiCardInterface – Wrapper to control wireless LAN card operations</td>
<td>46</td>
</tr>
<tr>
<td>4.11 Communication message format in OppCommFramework</td>
<td>47</td>
</tr>
<tr>
<td>4.12 Communication session break and restore algorithm</td>
<td>50</td>
</tr>
<tr>
<td>5.1 Time of Contact Distribution (Open Space)</td>
<td>56</td>
</tr>
<tr>
<td>5.2 Performance Analysis of Neighbor Selection Algorithms (Open Space)</td>
<td>57</td>
</tr>
<tr>
<td>5.3 Time of Contact Distribution (Shopping Mall)</td>
<td>58</td>
</tr>
<tr>
<td>5.4 Performance Analysis of Neighbor Selection Algorithms (Shopping Mall)</td>
<td>59</td>
</tr>
</tbody>
</table>
5.5 Time of Contact Distribution (Closed Lab Environment) .................................................. 60
5.6 Performance Analysis of Neighbor Selection Algorithms (Closed Lab Environment) ....... 61
5.7 Credibility tests results .................................................................................................. 62
LIST OF TABLES

4.1 Payoff matrix for prisoner’s dilemma................................................................. 48
CHAPTER 1
INTRODUCTION

In past decade, the number of mobile phone subscribers worldwide has increased by more than 1000%. [7] The current number of smartphone users, currently about 25 millions, is expected to increase four folds by 2013. [9] This explosion in the number of mobile/smartphone users and advanced capabilities of mobile devices like powerful processors, abundant memory, high screen resolution, camera and motion sensors, and continuous network connectivity; have given rise to wide range of smartphone applications. Using online web portals like iPhone AppStore [5], Android Market [3], and Windows Mobile Marketplace [10] users can download and install smartphone applications.

Currently most of the smartphone applications are either stand-alone or require internet connection. However, with increase in the density of smartphone users [7], another class of applications is emerging. These applications can communicate with neighboring devices to exchange useful information. The form of information exchange is generally user-controlled and manual i.e. user initiates and controls the communication with neighboring devices. Such applications include photo or file sharing, chatting, interactive games. Bluetooth and Wi-Fi are key communication technologies for these applications.

Consider a scenario in which mobile devices can communicate with each other, with minimal user supervision. Such applications will be immensely useful as background applications that collect useful information without user intervention and involve users only when it is necessary. For e.g. automatically broadcasting the list of user-requested files and
downloading them from multiple sources without user intervention or autonomous sharing of information with people of similar research interests, during research conference. [1]

Applications mentioned above come under the category of opportunistic networking applications. Most of the currently available opportunistic networking applications run on Bluetooth protocol. For example, MobilClique implements ad hoc social connections, epidemic newsgroups and asynchronous messaging using Bluetooth. [4] Mobile social networking application – Aka Aki finds members of a social network present in the neighborhood, using Bluetooth. [2]

While, Bluetooth is an effective low-energy wireless protocol for exchanging data over short distances; Ad hoc Wi-Fi provides much better communication range with higher data rates. Our testing mobile device – HP iPAQ 910c has up to 109-meter Wi-Fi range and approximately 10-meter Bluetooth range. Maximum data rate using Bluetooth 2.0/EDR is 3 Mbps as compared 54 mbps data rate achieved using Wi-Fi (802.11 b/g). [8] Also, device discovery using Bluetooth takes 10 to 17 seconds, as compared to 1-2 seconds for Wi-Fi device discovery. Almost all the smartphones nowadays come with built-in Wi-Fi card, making ad hoc Wi-Fi technology as available as Bluetooth. Only advantage of Bluetooth technology has over ad hoc Wi-Fi is that it uses fifth the power of Wi-Fi. [11] But if Wi-Fi can be utilized in energy-efficient manner, this difference can be compensated for.

Despite above mentioned advantages, the use of ad hoc Wi-Fi technology for developing opportunistic networking applications is limited. Also, existing Wi-Fi based applications like MultiCommFramework or iClouds do not completely satisfy the needs of opportunistic networks.

MultiCommFramework is an application library that enables .NET compact framework and .NET framework applications to exchange information, over ad hoc Wi-Fi.
MultiCommFramework works on the principle of multicasting, where one listener coordinates the communication between multiple clients. Clients send UDP message to listener, which is then multicast to other clients and listener multicasts UDP messages to clients as well. In this form of communication, at least one node should act as a listener i.e. if listener node shuts down, entire communication stops. Also, device discovery in MultiCommFramework is done using invitation message broadcast, which is not very efficient way publishing information and discovering devices. Communication in opportunistic networks mostly takes place between two neighbors in one hop manner. Thus, MultiCommFramework is not directly suitable for opportunistic networks. Since, nodes in opportunistic networks can come in contact unknown nodes, it is useful if nodes have a strategy to choose a particular neighbor. There exists no such mechanism in MultiCommFramework.

iClouds is another ad hoc Wi-Fi networking architecture that aims at distributing information in peer-to-peer manner. iClouds uses innovative push-pull mechanism to get requested information and share available information over the network i.e. it uses iHave list to indicate the list of locally available information resources, and iWish list to indicate the requested information resources. These lists are either sent to or requested from neighboring devices. iHave list of a neighboring device is matched against device’s iWish list and requested information is retrieved. iClouds architecture is used for two purposes: gathering data autonomously and notifying its users about presence of suitable neighbor in the surroundings. This notification is in the form of beep or vibration. After getting notification, further communication the users take charge of communication. Thus iClouds supports only exchange of preliminary information. The architecture doesn’t specify any model for advanced form of communication between two devices. For example, two devices might want autonomously exchange series of messages in synchronous fashion. Also, iClouds doesn’t try to establish credibility or information contribution of its neighbors. This enables possibility of nodes with no information to share, in the network. These nodes are called ‘freeriders’ as they only extract
information from the network without contributing anything. Also, there is no mechanism to validate the credibility of nodes in iClouds network. Thus, there is possibility of continued presence of nodes distributing spurious information in the network, without any mechanism to find how credible they are.

We design and implement an application architecture using ad hoc Wi-Fi technology that addresses the needs of opportunistic networks. If we want to build an opportunistic networking application running on smartphones, timely discovery of neighboring devices is expected. Since opportunistic networks rely on proximity of neighbors and are highly mobile in nature, speedy discovery will ensure neighboring devices are detected before they go out of range. Use of ad Wi-Fi over Bluetooth technology ensures speedy discovery, its longer range provides more contact opportunities and longer opportunistic contact period. We use beacon stuffing technique publish application and device information, thus using MAC layer device discovery technique for neighbor discovery. This enables faster and energy-efficient neighbor discovery.

Another important issue in opportunistic networks is selecting best neighbor for communication. If an attempt to connect to a particular neighbor fails even after multiple tries, it might be best idea not to communicate with that neighbor. Even when multiple neighbors are found in the surroundings, all neighbors might not be equally suitable for communication. For example, a node found during device discovery might be on the verge of leaving communication range or a node might be relatively mobile as compared to other nodes. Thus, an attempt should be made to choose a neighbor carefully to ensure further communication goes smooth. Neighbor selection problem is even more important in case of applications exchanging conversation messages, instead of simple sharing of data. While exchanging shared data, if communication is broken against one neighbor, remaining data could possibly be retrieved
from other neighbors. In case of conversing applications, broken session with one neighbor can only be reinitiated with same neighbor.

Thus, we have come up different mechanisms based on signal strength values and time of contact between neighbors, which assist nodes in choosing best neighbors out of all available neighbors. Also, ensuring that the neighbor is credible enough before exchanging information is critical. We assign credibility to neighbors at the end of every successful or unsuccessful information exchange, thus helping nodes choose reliable neighbors during future encounters.

Using neighbor selection strategies, we ensure that neighbor stays in the range of node when communication is in progress. Once a suitable neighbor is found, next step is to establish connection with it and handle message exchange between nodes. As nodes in opportunistic networks depend on user mobility and proximity, communication may break if we might expect frequent breaks in communication. So, opportunistic networking architecture should have some mechanism to handle communication breaks, if possible restore broken communication. We provide an enhanced version of MultiCommFramework that is more suitable to opportunistic networks i.e. one hop message exchange between two neighbors at a time.

We have implemented this architecture on HP iPAQ 910 mobile devices running on Window Mobile 6 operating system. Apart from .NET compact framework 2.0, we make use of smart device framework 2.3 developed by OpenNETCF consulting. This framework provides basic functionality to control radio operations and wireless LAN card on Windows Mobile device.

We performed series of experiments in environments ranging from open space to shopping malls to in-house lab environment and analyzed the performance of various neighbor selection strategies, based on RSSI and time of contact values. A strategy is favored if a selected neighbor stays in surroundings for longer time. We will discuss how these neighbor
selection strategies and publishing application information using Wi-Fi beacon stuffing provides faster neighbor discovery and more communication opportunities in energy-efficient manner.

With growing interest in smartphone applications and increasing numbers of smartphone users, opportunistic communication between neighboring users is becoming more and more apparent. People shopping at malls, meeting in social functions or simply meeting on street provide numerous possibilities to share information and communicate with neighboring devices. It is this wide range of possibilities that inspired us to design application architecture for communication and information exchange among smartphone users, using commonly available ad Wi-Fi technology.
CHAPTER 2
BACKGROUND

In following sections we will understand opportunistic networks (Chapter 2.1), wireless LAN protocol (Chapter 2.2).

2.1 Opportunistic Networks

Consider a scenario where a recent graduate student is at a career fair looking for job positions suitable to his/her skillset. Typically a student will go to every company’s booth and get to know the company and tell recruiters about his/her interests. This is very time consuming process and student might miss out on many good companies because of lack of time or prior information about company. Also, companies might miss out on some very good candidates just because students didn’t know that a particular company is looking for people with his/her skillset. This is where opportunistic networks can be very useful. If company recruiters have PDAs equipped with Bluetooth or Wi-Fi, they can broadcast information about open job positions and contact information. This way all applicants will get information about companies and what skills they are looking for without actually having to meet recruiters. Also, if students can broadcast important resume keywords and contact information from their mobile phones, recruiters can easily get to know the candidates that are best suited for open job positions. In fact, if there is a strong match between open job position and student’s skillset, recruiter can immediately reach a candidate via phone and initiate interaction. Also, student mobile phones can contact their friends’ mobile phones, and suggest them a particular company.

In above example, there were two important factors in communication. One, finding suitable devices in neighborhood and second, exchanging information between these devices,
both without much user intervention. Opportunistic networks make perfect case for this type of communication i.e. opportunistic networks make people physically aware of each other’s presence and exchange useful information without user’s knowledge.

Opportunistic network is defined as network of mobile or fixed wireless connected nodes. This network may change its topology due to activation or deactivation or mobility of nodes. Communication between these nodes takes place within walking distances (~100-300 meters). [12]

These nodes at least provide following functionalities:

1) Node discovery: Node in a network can detect other nodes in direct communication range.

2) One-hop message exchange: Node can send or receive arbitrary data from surrounding nodes that are in direct communication range.

Opportunistic networks can contain two types of nodes, first, mobile nodes and second, fixed nodes also known as information sprinklers. A mobile node consists of a user carrying mobile device with short range wireless communication capabilities. Information sprinklers are fixed devices, with no user control. They can act as a sprinkler and disperse information, or as a sink and collect information from mobile nodes, or as both at the same time. Information sprinklers can form network of their own, with other information sprinklers over a backbone network. Although information sprinklers are optional nodes, a typical opportunistic network looks as follows:
In Figure 2.1, we see mobile nodes spread around the network. They form connection links (dotted lines) with other mobile nodes and information sprinklers that are within their direct communication range (dashed circles). Information sprinklers might be connected to each other over a backbone network (dash-dotted lines).

It is important to understand the differences between opportunistic networks and other self-organizing networks - mobile ad hoc networks (MANETs) and peer-to-peer networks (P2P).

Mobile ad hoc networks are formed by a network of mobile and self-configuring nodes that act as routers for communication between other nodes. MANETs have rapidly changing and unpredictable wireless topology, but all nodes trust each other and share a common goal to accomplish. In contrast, opportunistic networks are formed between unrelated nodes and anonymous therefore unknown users. Message exchanges in opportunistic networks are mostly one-hop and they lack message routing component present in MANET nodes. Also, MANETs reside on network layer, whereas opportunistic networks reside on application layer.

Peer-to-peer network (P2P) is a distributed network architecture where participating nodes act as resource providers or resource requesters for service and content. A node in
opportunistic network also acts as information requester and provider. P2P and opportunistic networks are both formed at application layer. But in internet P2P network, mobility is absent and its size is several orders of magnitudes higher than typical opportunistic networks. Mobile P2P networks have node mobility, but they heavily depend on request forwarding and message routing. In contrast, nodes in opportunistic networks communicate using one-hop message exchanges.

Following concepts will make understanding of opportunistic network fluid: [12]

a) Exploitation of user proximity: People present at same place and time give rise to possibility of short range communication, which might create an opportunity to meet face-to-face with each other. This might be fruitful in some cases, such as two unknown people present at jazz concert having common favorite artists or researchers gathered at research conference having common research interests. Even otherwise, user proximity can help user meet new people.

b) Expressing user interests based on profiles: Discovering nearby devices, thus users is not very useful if there is no way to find out whether further communication will be fruitful or not. This is generally achieved by placing user profile on device describing his/her interests concisely. This user profile can contain user information or knowledge to be shared.

c) Data dissemination: If users A and B have similar interests and user A has some knowledge which can be shared with user B, it is transferred from A’s device to B’s device. Data dissemination process depends on number of nearby users with similar interests and user mobility, which physically carries data from one place to other.

d) Open network and unrelated group of users: Usually users in opportunistic networks are unknown to each other and independently communicate with other nodes, may be out of self-interest.
e) Lack of predictability in communication pattern: Opportunistic networks can at most provide ‘best efforts’ service, making some applications unsuitable to run on opportunistic networks i.e. user cannot entirely rely on opportunistic networks to satisfy his/her interest. For example, researchers gathered at conference might still miss out on each other because they were out of communication range, despite being present at same place and time.

Based on above mentioned ‘best efforts’ property, opportunistic networking applications can be categorized in two types:

1. Active Collaboration: These applications not only exchange digital information with nearby users by exploiting physical proximity, but also act as link to users. For example, in the career fair example at the beginning of this chapter, if recruiter’s device finds a strong match between job position and candidate profile, it might alert recruiter by some nonintrusive intimation like vibration. Recruiter can then contact the candidate with strong profile. Actively collaborating applications do not store complete user information, but just enough to take communication between two parties to next level.

   Some existing examples of Active Collaboration applications are Lovegety – a small mobile to get people introduced with each other [13], and Nokia Sensor – Bluetooth application for Nokia mobile phones [14]. In Lovegety, a device that can be set in one of three states finds neighborhood devices within five meter range. When two neighboring devices are found to be set to same state, both devices beep and inform holders to search each other physically. In Nokia Sensor, users can set their profiles and store their interests like music taste, on Bluetooth enabled mobile phones. Application lets users search each others’ profiles within walking distance and forms on-the-fly communities. Moreover, users can choose to share files with each other.
2. Passive Collaboration: These applications focus on data dissemination without user interaction. Passive Collaboration applications autonomously interact with each other spreading information using word-of-mouth communication. Since private resources like battery, memory, or processor power are shared without user's knowledge, there should be some incentives to make user want to participate in data dissemination. Second important issue in Passive Collaboration applications is privacy protection mechanisms.

Some existing examples of Passive Collaboration applications are autonomous gossiping for mobile and iClouds. [15] In iClouds, every device has two information lists (iLists):

a. iHave-list (information have list): This list specifies information node wants to shared with other nodes.

b. iWish-list (information wish list): This list specifies information node is interested in.

Due to heavy human involvement in opportunistic networks, privacy and incentives become important issues.

a. Privacy: User privacy might be breached if user can be identified based on information collected through opportunistic communication. Users can choose to use their own identity or a pseudonym or can choose to remain completely anonymous.

b. Incentives: Since private resources are shared in opportunistic networks, there should be some incentive for users to participate in it. A common benefit is satisfaction of user's information needs. Otherwise, a point scheme can be implemented to encourage users to participate in communication.
2.2 Wireless LAN Protocol

Fundamental difference between wired and wireless network is that in wired network station address also implies its physical location. In wireless networks, stations (STA) are assumed to be mobile and portable. Also, since mobile stations might be battery powered, wireless LAN protocols must take into account power management.

IEEE 802.11 defines a set of standards to carry out wireless local area network communication at physical (PHY) and medium access control (MAC) layer and Wi-Fi Alliance, a trade group certifies wireless devices based on IEEE 802.11 standards and guarantees interoperability between different wireless devices.

2.2.1 IEEE 802.11 Protocol Architecture

At physical layer, IEEE 802.11 defines two sublayers:

a. Physical Layer Convergence Procedure (PLCP): This sublayer defines specifications for converting MAC Layer Protocol Data Units (MPDUs) into suitable framing format. This enables sending and receiving of user data and management information between two or more STAs using underlying PMD sublayer.

b. Physical Medium Dependent Sublayer (PMD): This sublayer defines specifications for methods of transmitting and receiving user data and over wireless medium between two or more STAs and characteristics of user data.

Above the physical layer, medium access control layer (MAC) and logical link control layer (LLC) provide following functionalities: [17]

1. On transmission, assembly of data into frames with address and error detection fields.
2. On reception, disassembly of data frames, error detection and address recognition.
3. Access control to wireless LAN medium.
4. Provide an interface to higher layers and perform flow and error control.

Figure 2.2 shows main components of typical IEEE 802.11 network.

Basic Service Set (BSS) forms the basic building block of wireless LAN. It consists of number of STAs over same MAC protocol and competing for shared wireless medium. Basic Service Area (BSA) is the coverage area within which member STAs stay in communication with each other. When an STA moves out of this area, it can no longer communicate with other APs in this area. BSSs can be isolated or can be connected to backbone distribution system (DS) through access point (AP). Access point is a relay point between BSS and distribution system i.e. MAC frame addressed for an STA in another BSS is relayed by AP to distribution system. AP also acts as a bridge between two STAs of same BSS.
Independent Basic Service Set (IBSS) is the most basic type of IEEE 802.11 LAN. It can contain only two STAs at minimum. In this mode, STAs communicate with each other directly. Networks formed using this mode known as ad hoc networks.

When two or more basic service sets are connected to each via distribution system, it is called extended service set (ESS). To integrate wireless LAN architecture with other wired LANs portal is used. Portal is generally a bridge or a router that connects wired LAN and distribution system.

IEEE 802.11 specifications describe nine services. [17] They are as follows:

- **Association**: In order to communicate with other STAs, an STA associates itself with an AP within a BSS. Association gives identity and address to an STA.
- **Reassociation**: When STA moves away from one BSS and toward another BSS, established association is transferred from one AP to another.
- **Disassociation**: An AP or a station can send a termination notification indicating that it is leaving an ESS or shutting down. Sometimes such notification is not received, in which case MAC layer recognizes disappearing station on its own.
- **Authentication**: IEEE 802.11 requires mutually acceptable and successful authentication before a station can be associated with an AP.
- **Deauthentication**: IEEE 802.11 terminates existing authentication when this service is terminated.
- **Privacy**: IEEE 802.11 supports optional use of encryption schemes to support privacy of the data being transmitted.
- **Distribution**: If MAC frames are to be exchanged between two STAs that lie two different BSSs, AP of source STA transfers frames to DS and DS directs them to an AP of destination STA. Distribution service takes care of MAC frame traversal within DS.
• Integration: Data transfer between a station in wireless LAN and a station on an integrated 802.x LAN is managed by integration service i.e. integration service handles media conversion logic and address translation required for such data exchange.

• MSDU Delivery: MAC Layer Data Units - MSDUs are passed from MAC user to MAC layer to send out to destination address. Delivery service handles delivery of MSDUs between stations. If MSDU is too large, it is fragmented and is transmitted in a series of MAC frames.

2.2.2 IEEE 802.11 Medium Access Control Layer

IEEE 802.11 MAC layer specifies three functionalities: reliable data delivery, medium access control and security.

2.2.2.1 Reliable Data Delivery

IEEE 802.11 provides two mechanisms to ensure reliable frame delivery. In frame exchange protocol, for every frame received, an acknowledgement (ACK) frame is returned to source. If receiver doesn’t receive a frame or if ACK frame send by receiver is lost, source sends concerned frame again.

For enhanced reliability, four-frame exchange mechanism is used. In this mechanism, a source issues request to send (RTS) frame to the destination. This frame alerts nearby stations to refrain from transmission so that collisions can be avoided. Destination sends back clear to send (CTS) frame to source. On receiving CTS frame, source transmits data and destination replies with ACK frame. Although this mechanism ensures enhanced reliability, it can starve other stations from transmitting data. Hence, this mechanism is disabled by default.

2.2.2.2 Medium Access Control

IEEE 802.11 working group considered distributed and centralized approaches for medium access control. In distributed access control, decision to transmit is taken using carrier-
sense mechanism. Whereas in centralized access control, a centralized decision maker controls the access mechanism. In order to satisfy both these approaches, the working group came up with distributed foundation wireless MAC (DFWMAC) algorithm that provides distributed access control mechanisms as well as optional centralized control on a layer above it.

![IEEE 802.11 Protocol Architecture](image)

Figure 2.3: IEEE 802.11 Protocol Architecture [17]

As seen in figure 2.3, distributed access control is handled by distributed coordination function (DCF) sublayer and centralized access control is managed by point coordination function (PCF) sublayer.

DCF sublayer uses carrier sense multiple access (CSMA) mechanism for transmitting MAC frames as follows:

1. Before transmitting a frame station senses the medium. If the medium is idle, it waits for Interframe Space (IFS) time period. If medium remains idle during this period, station transmits immediately.
2. If the medium is busy, station postpones transmission and continuously monitors the medium until it becomes idle again.
3. When current transmission is over, station waits for another IFS time period. If the medium remains idle during this period, station backs off by a random amount of time and senses the medium again. If the medium is still idle station transmits. Backoff timer is stopped if medium becomes busy during backoff period and it resumes when the medium becomes idle again.

4. If after transmission acknowledgement is not received, it is assumed that collision has occurred.

In PCF sublayer, point coordinator polls all stations configured for polling in round-robin fashion. If a station has a frame to send, it transmits it when polled by point coordinator. Since IFS time period for point coordinator (PIFS) is less than the IFS time period for DCF sublayer (DIFS), point coordinator gets higher priority while accessing the medium.

2.2.3 IEEE 802.11 Physical Layer

IEEE 802.11 working group has issued physical layer in four stages. First stage, IEEE 802.11 specifies medium access control layer and following three physical media [17]:

- Direct Sequence Spread Spectrum (DSSS): This system operates in 2.4 GHz ISM band with data rates of 1 and 2 Mbps. Up to three non overlapping channels each with bandwidth of 5 MHz is used. Differential binary phase shift keying encoding scheme is used.

- Frequency Hopping Spread Spectrum (FHSS): This system operates in 2.4 GHz ISM band with data rates of 1 and 2 Mbps. FHSS system hops between multiple channels based on pseudonoise sequence.

- Infrared: It is an omnidirectional scheme with range of up to 20 meters. It uses 16-Pulse Position Modulation encoding for data rate of 1 Mbps.
IEEE 802.11a operates in 5-GHz band with orthogonal frequency division multiplexing (OFDM) scheme. This scheme uses multiple carrier frequencies each to transmit some data bits from single data source. IEEE 802.11a provides data rates up to 54 Mbps.

IEEE 802.11b extends IEEE 802.11 DSSS scheme to provide data rates of 5.5 and 11 Mbps in 2.4 GHz ISM band. IEEE 802.11g is a successor of IEEE 802.11b providing data rates up to 54 Mbps. IEEE 802.11g also works in 2.4 GHz range.

Recently IEEE 802.11 working group approved IEEE 802.11n standard that provides data rates up to 600 Mbps, surpassing maximum data rates offered by all previous standards.

2.2.4 IEEE 802.11 in ad hoc mode

As briefly mentioned in previous content, independent BSS (IBSS) forms the most basic type of IEEE 802.11 LAN. It may consist of only two STAs, which communicate with each other directly. Since these networks are formed without any pre-planning, they are termed as ad hoc networks. One of the STAs takes responsibilities of access point. It handles periodic beaconing and authentication of new member STAs. But it does not act as a bridge to relay transmission between STAs and STAs communicate directly with each other. There can be maximum nine members in an IBSS network. As seen in figure 2.4, independent BSSs can coexist independently, even overlap with other IBSSs or ESSs.
Since STAs in IBSS network directly communicate with each other, there is no distribution system, hence no portal or integrated wired LAN. In IBSS, each STA enforces its own security policy, as against AP enforcing security and access mechanism in BSS network.
CHAPTER 3
RELATED WORK

3.1 Beacon Stuffing

There are number of similarities between wired networks and IEEE 802.11 wireless network. Two nodes in a wired LAN can communicate with each other only when they are connected to an Ethernet bridge or switch. In wireless network, an STA should be associated with an access point (AP) to communicate with other STAs on same AP. Although STA associated with an AP can hear beacons from other APs in the neighborhood, this model puts restriction on communication between STAs associated with different APs.

One obvious solution to above mentioned problem is to use multiple wireless cards, so that each card can connect to different AP and an STA can communicate over multiple APs. However, this will result in excessive energy drain and reduced battery life of mobile phones. Some attempts have been made to allow STA to communicate with multiple APs using single wireless card. These attempts involve inclusion of an intermediate layer between IP and MAC layer. [18] Since this approach requires making significant changes to network protocol stack, this approach might not be very desirable.

Ranveer Chandra, Jitendra Padhye and Alec Wolman at Microsoft Research have come up with an innovative mechanism to communicate with STAs using a low bandwidth protocol named ‘beacon stuffing’. This mechanism requires minimal wireless card driver changes and some of the mechanisms work even without making any driver changes. These protocols have been tested on multiple wireless cards and Windows XP and Windows Vista operating systems.
Basic idea of this protocol is to embed useful information like advertisement of network services or real world goods or location-specific ads inside IEEE 802.11 management frames like beacons, probe requests and probe response. The protocol uses push model for information delivery. APs broadcast information overloaded IEEE 802.11 beacons to notify network presence. Since STAs are capable of discovering beacons from all nearby APs, when they scan the network, they detect presence of information overloaded AP beacons. Typically short text messages are exchanged, but short audio tunes can also be exchanged. A long message is split in multiple parts and each fragment is sent on separate beacons. In case of multi-part message, each fragment has following format:

<table>
<thead>
<tr>
<th>Unique ID</th>
<th>Sequence Number</th>
<th>MoreFlag</th>
<th>InfoChunk</th>
</tr>
</thead>
</table>

Figure 3.1: Message fragment in beacon stuffing protocol [19]

*UniqueID* field is an identifier for a broadcasted message; *SequenceNumber* stores the fragment number and *MoreFlag* indicates presence of more fragments. *MoreFlag* contains value of 0 for last message fragment and value of 1 for all fragments before the last one. *InfoChunk* contains the information sent as a part of message fragment.

Some important fields in 802.11 beacon frame are as follows:

| Timestamp (8 bytes) | Beacon Interval (2 bytes) | Capability (2 bytes) | Service Set Identifier (32 bytes) | Supported Rates (8 bytes) | Information Element (256 bytes) | BSSID (6 bytes) |

Figure 3.2: Important fields in IEEE 802.11 beacon packet [19]

Multiple techniques are used to embed data in 802.11 beacon packets. These techniques differ based on which field in 802.11 beacon packet is used to carry data. [19]
1. **SSID Concatenation**: In this technique, SSID field in beacon packets is used to carry information. SSID field can contain a maximum of 32 bytes. If `UniqueID` field in message fragment (figure 3.1) is assumed to be 1 byte and `SequenceNumber`, `MoreFlag` fields are fit in 1 byte; maximum 29 bytes remain for `InfoChunk` field of message fragment. Maximum message length can be 3712 bytes and message fragments can be transmitted in successive beacons. If beacon interval is set to 10ms, maximum message transmission speed of 23 Kbps can be achieved. SSID concatenation is the easiest technique to implement as SSID field can be easily modified on all commercial access points. Also, clients can query from user level to get SSID field information in beacon packets of nearby APs. Thus, this technique can be implemented without doing any modification in access points and client devices. A problem with this approach is it creates a large number of bogus SSIDs.

2. **BSSID Concatenation**: Similar to SSID field, BSSID field in beacon packets can be used to transmit message fragments. BSSID field can store 6 bytes of data. If 1 byte is used for `UniqueID` field and 1 more byte is used for `SequenceNumber`, `MoreFlag` fields combined; remaining 4 bytes can be used for transmitting data. This enables transmitting a maximum of 512-byte message at the rate of 3.2 Kbps, if beacons are sent every 10ms. Since clients only store a list of unique SSIDs, this approach doesn’t swamp clients with bogus SSIDs.

3. **Beacon Information Element**: IEEE 802.11 protocol allows 253 bytes of beacon information element (BIE) field in beacon packets to be filled with vendor-specific information. Storing `UniqueID`, `SequenceNumber` and `MoreFlag` fields in 2 bytes (like previous techniques), 251 bytes can be used for sending messages. This approach yields a high data rate of 200 Kbps, but it comparatively difficult to implement. Modification in client driver is required to be able to read BIE field in beacon packets.
There are numerous applications of beacon stuffing techniques. Some of them are as follows: [19]

1. Network Selection: In general, wireless zero configuration service on Windows allows users to choose 802.11 wireless access point based on network name and signal strength. Using above mentioned techniques, beacons can be overloaded with additional information that will be helpful to end-users in making better decision while selecting a network. This information can be name of the service provider, pricing information, whether network is available for public or private use etc.

2. Wi-Fi advertisements: Consider a scenario where customers are roaming in a shopping mall and they receive new product notifications or discount sale information on their mobile phones automatically, while passing by their favorite outlets. Such location-sensitive ads can be implemented based on beacon stuffing techniques. Also, since maintaining on-the-go internet connection is costly and difficult; above mentioned techniques make perfect case for targeted ads. Using push delivery model, 802.11 clients can receive advertisements from nearby APs, whether or not they are associated with those APs. Another advantage of beacon stuffing techniques is their implicit location awareness. This enables overwhelming users with irrelevant advertisements as in the case of internet.

3. Coupon Distribution over Wi-Fi: Coupon distribution is very useful technique for promoting new products and improving sales of existing ones. Consider a scenario in which a smartphone user is passing by a coffee shop that is broadcasting discount coupons for newly introduced coffee flavors. When user receives a coupon over Wi-Fi, he or she can choose to save it and utilize it. Using Ad Service Provider cryptographic techniques [19], distribution and identity of coupons can be protected.

Beacon stuffing protocols have been implemented on Windows XP and Windows Vista operating systems. AP functionality is implemented on Realtek 8185 and Atheros 5523
chipsets, whereas client functionality is tested on wireless cards from 5 different vendors. On client side, application periodically queries driver for beacons with embedded information and assembles message fragments from multiple beacons.

In given implementation, there is implicit assumption of separation between AP and client devices, thus ignoring the use of beacon stuffing in 802.11 ad hoc mode. SSID field in beacon packets can be used to transmit and receive useful user level information on mobile devices.

3.2 MultiCommFramework

As the density of smartphone users is increasing, need to share information among peer devices is increasing. Mere stand-alone applications are not sufficient and smartphone applications now communicate with desktops and nearby mobile device at real-time. Since mobile devices mostly communicate over wireless medium, and wireless medium is not as reliable as wired medium; new breed of smartphone applications should take intermittent network connectivity into consideration.

MultiCommFramework is one such application library that enables .NET compact framework and .NET framework applications to exchange information, over ad hoc Wi-Fi network. Conceptually MultiCommFramework is similar to multicasting, wherein client applications register with network router for retrieving information of their interest. Presence of explicit network router is absent in case of ad hoc networks, hence one device runs listener or host application, which acts similar to router. This application registers other devices as clients and distributes messages among them. The host application can also act as a client and send messages to other clients. As MultiCommFramework is intended for loosely connected networks, it uses UDP datagrams for communication.

In order to use MultiCommFramework, applications decide on two ports for communication, and a session or conversation identifier. One port is used by clients to listen to incoming messages and other port is used by listeners to receive information from clients. The
application library has two main components: MultiCommListener and MultiCommClient. When application initializes, it can choose to be either of the two. In general, we require only one MultiCommListener application to connect to clients and distribute messages, and one or more MultiCommClient applications.

Following are the steps followed by MultiCommClient application:

1. Initialize MultiCommClient component by calling constructor for MultiCommClient class, sending conversation identifier and communication ports.
2. Create event handling functions for events generated by MultiCommFramework.
3. Call method start on MultiCommClient. This launches background thread for listening to incoming requests.
4. MultiCommListener application sends out messages to invite clients. When MultiCommClient application receives such invitation, it connects to listener on the IP address and port sent through the message.
5. Alternately, MultiCommClient application can broadcast invitation request message using SendInviteRequest method. If listener is present in vicinity, it responds with invites client to join conversation.
6. After connecting with listener, client can send messages to other clients and receives messages from them.

Following steps are followed by MultiCommListener application:

1. Initialize MultiCommListener component by calling constructor for MultiCommListener class, sending conversation identifier and communication ports.
2. Create event handling functions for events generated by MultiCommFramework.
3. Call method start on MultiCommListener. This launches two background threads: one for listening incoming messages from clients and other for distributing messages among clients.
4. *MultiCommListener* application broadcasts invitation message using *SendInvitation* method. This helps clients in identifying the listener IP and port, ultimately allows them to connect to listener, hence other clients.

5. Once connected with clients, listener has similar functionality of receiving messages from clients and multi-cast messages to all clients.

*MultiCommFramework* fires events to inform applications of certain activity taken place internally. Following is the description of events and explanation on how they should be handled.

1. **OnReceive**: This is the most critical event fired by *MultiCommFramework* and it informs application about arrival of message from client. Clients call *Send* method to send messages to other clients. When these messages are received by other clients, *OnReceive* event is fired.

2. **OnJoin**: This is an informational event informs listener about joining of new client. Also, this event is fired on the newly joined client itself. Listener receives conversation identifier and IP address of the newly joined client, and new client receives IP address and conversation identifier of listener.

3. **OnLeave**: Similar to **OnJoin** event, this event signifies leaving of a client. This event is fired on both listener and leaving client. Listener receives IP address and conversation identifier of the client just left. Leaving client receives IP address and conversation identifier of a listener.

4. **OnTerminate**: This indicates that listener is leaving the conversation and clients will not be able to communicate with each other. This event only stores the conversation identifier.

*MultiCommFramework* is designed to handle all network communication issues with minimal overhead and let application developers focus on application development.
MultiCommClient and MultiCommListener classes are created based on inheritance relationship. MultiCommClient represents a base class, which is extended by MultiCommListener class. MultiCommListener adds or overrides little functionality of its own representing minimal code overhead, and uses most of the MultiCommClient code.

MultiCommClient mainly sends and receives messages on behalf of the application. ReceiveMessagePump method handles reception of incoming messages on a background thread. Once message is received, it hands it over to ReceiveDispatchMessage method for processing.

ReceiveDispatchMessage is a virtual method with client and listener specific behavior. For a client, processing is done using a finite state machine with following states:

1. Waiting for Invitation: Client application is running and is waiting for invitation from listener.
2. Waiting for Invitation Acknowledgement: Client has received invitation to join conversation from listener, and has sent a message to listener to send an acknowledgement. If acknowledgement is not received, client rolls back to previous state – Waiting for Invitation.
3. In a Conversation: Client is connected with listener; it is sending and receiving conversation messages. At this stage, client can either receive more conversation messages or a termination message from listener. If conversation termination message is received, client roles back to Waiting for Invitation state.

MultiCommListener inherits background thread handling part from MultiCommClient; it overrides the implementation of ReceiveDispatchMessage method. In this overridden method, listener handles accepting invitations for new clients, receiving requests for invitation, conversation message and request to leave conversation. MultiCommListener also handles message distribution between clients, which is implemented by running another background
thread encapsulating `DistributeMessagePump` method. This method implements a message queue that allows listener to broadcast a message or send it to a specific client.

Following are different message types implemented in `MultiCommFramework`. [20]

1. **OpCodes.Invite**: This message is broadcasted by listener when `SendInvitation` method is called. Alternately, this message can be sent to a specific client if `OpCodes.InvitationRequest` message is received from it.

2. **OpCodes.InviteRequest**: Client broadcasts this message by calling `SendInvitationRequest` method. This informs listener that new client in vicinity wants to join conversation.

3. **OpCodes.InviteAccept**: When client receives `OpCodes.Invite` message, it responds with `OpCodes.InviteAccept` message. Listener registers new client, so that it can also receive conversation messages. This message triggers `OnJoin` event on listener and specific client.

4. **OpCodes.ConversationBody**: This message is sent by client to listener by calling `Send` method. Listener puts this message in queue to be forwarded to each of the registered clients. This message triggers `OnReceive` event on clients and listener.

5. **OpCodes.LeaveConversation**: When client no longer wishes to receive conversation messages, it sends out this message to listener. Listener then unregisters client from the list of registered clients. This message triggers `OnLeave` event on listener and concerned client.

6. **OpCodes.TerminateConversation**: This message is sent by listener to inform termination of services provided by it. This message triggers `OnTerminate` on listener and all the clients.

In a nutshell, `MultiCommFramework` provides framework for building multicasting applications on Windows Mobile platform using ad hoc Wi-Fi technology. Applications built on top of `MultiCommFramework`, receive information about network activities via events fired by
the framework. However, *MultiCommFramework* doesn’t focus on the need of opportunistic networks i.e. device discovery is done using invitation messages, which is time consuming and might involve redundant invitation message broadcasts. Also, *MultiCommFramework* focuses on multicasting between number of clients and a listener, and might not ideal for two-party communication in opportunistic networks. Neighbor selection is another important problem in opportunistic networks, which is not answered by *MultiCommFramework*.

### 3.3 iClouds

iClouds is an ad hoc peer-to-peer architecture developed by department of computer science at Darmstadt University of Technology. The motivation behind this architecture is if people are gathered in a group, they might have a common information goal. iClouds aims to provide this information to the group in peer-to-peer manner.

An iCloud or information cloud is a communication range within which a person carrying wireless-enabled PDA can be contacted. Since iCloud has limited communication range, only the people in vicinity can communicate with such iCloud. Hence, limited communication range becomes a desirable quality.

iClouds works on PDAs with IEEE 802.11 b support. Since it uses ad hoc Wi-Fi protocol for communication, maximum communication range is 100m. Communication between two iClouds nodes can happen in one hop manner only, hence support for routing protocols is ruled out.

Every iClouds node holds two data structures – iHave and iWish lists. [15]

1. **iHave list**: This list holds information that user wants to contribute to iCloud. Items in this list can be declared in simple string or XML format.
2. **iWish list**: User requested information is stored in this list. Typically this list contains search patterns that can be matched against iHave lists of neighboring nodes. iWish lists are more private compared to iHave lists. This gives users control over information they want to share with neighboring nodes.
iClouds device scans its neighborhood periodically. Record of newly found and known neighbors is stored in a data structure called ‘neighborhood’. After discovering each other, devices exchange their iLists. iWish list is matched against iHave list of a neighboring device. If wished item is found on neighbor’s iHave list, it is transferred to device’s own iHave list. iLists can be exchanged in two ways – push and pull. In push method, device pushes its iLists to its neighbors. In pull method, devices pull iLists from other devices. Both operations can be performed on iWish and iHave lists. This gives four different types of iList exchanges, as noted below: [15]

1. Standard Search: User A asks for iHave list of neighboring user B. It matches the list against its iWish list.

2. Advertise: User A pushes its iHave list on user B. This approach is used for actively publishing user owned information.

3. Active Service Inquiry: User A pulls user B’s iWish list. In this method, user A tries to find information on user B’s behalf.

4. Active Search: User A pushes its iWish list to user B. In this method, user A is actively inquiring about particular item.

Search patterns in iWish list and information items in iHave list are modeled in hierarchical manner. iWish list contains different information categories. iHave list stores available information in XML format, specifying various categories for each information node. Different categories are matched against category listings of every information node in iHave list and matching nodes form part of the result.

An iClouds device user can optionally set notifications when an information item is matched against its iWish list. This notification can be in the form of beep or vibration. The purpose of setting notification is to intimate user to initiate further communication with neighboring node. For example, after finding match against iWish list, user might want to
interact with his/her neighbor via call or actual conversation. If notifications are turned off, device might just be doing data collection.

iClouds prototype has been implemented on Toshiba Pocket PC e740 devices, which run on Windows CE platform. Applications are developed on Java ME platform. Application uses UDP ping-pong for scanning neighboring nodes. A scanning node periodically send ping message. When neighboring node replies with pong message, it is added to ‘neighborhood’ data structure. If a particular neighbor doesn’t reply for certain time limit, it is removed from ‘neighborhood’ data structure. Node makes HTTP GET requests to get iLists from neighbors.

In conclusion, iClouds provides useful platform for communication between mobile nodes in opportunistic setting, using ad hoc Wi-Fi connection. The push-pull model provides multiple ways to exchange information lists – iLists. But iClouds limits itself to exchange of preliminary information stored in iLists and leaves further communication to device users. This model is inadequate for establishing advanced form of communication like conversation or synchronous exchange of series of messages i.e. after gathering information from neighboring devices, users might want to initiate session to exchange messages in synchronous request-response fashion. There is no mechanism in iClouds to determine contribution of its neighbors. For example, a particular iClouds node may be ‘freeriding’ i.e. it contributes no information, but consumes all the information from its neighbors. Also, there is no mechanism to establish credibility of neighbors with whom information is exchanged. There is no way to know whether information exchanged is valid or not.
CHAPTER 4
APPLICATION ARCHITECTURE

4.1 Problem Specification

Opportunistic networks are formed based on the need to share information autonomously between mobile devices that are in vicinity of each other. Although users are aware of this information exchange, minimum user intervention is required to perform communication autonomously. Also, mobile nodes in opportunistic networks are loosely connected; hence frequent communication breaks are expected and should be handled effectively.

Consider a scenario in which mobile nodes in opportunistic network wish to exchange messages in synchronous manner. First, mobile device should perform device discovery to find its neighbors. Then, it should connect to one of the neighbors and begin message exchange. Since opportunistic networks are highly dynamic in nature, an attempt should be made to connect to a reliable neighboring device, which will ensure longevity and stability of communication. Communication can break if one device goes out of direct communication range of other device. Since message exchange in opportunistic network takes place in one-hop manner, no attempt is made to route a message through other devices in range. During such communication breaks, it is vital to store the communication state information. If a particular device comes back into communication range in future, an attempt should be made to reinitialize the broken communication. Also, it is important to lower the probability of communication with unreliable neighbors.

There are some existing applications that come close to satisfying the needs of above mentioned applications partially. For example, MultiCommFramework provides mechanisms for
establishing asynchronous communication between Windows Mobile smartphones using ad hoc Wi-Fi technology. Existing applications built on MultiCommFramework involve heavy user intervention. Also, MultiCommFramework is built based on the concept of multicasting messages to a group of nodes and node discovery mechanism involves joining a group of nodes before communication. These factors make MultiCommFramework unsuitable for opportunistic networks. However, message exchange mechanism in MultiCommFramework can be very useful in building framework for synchronous message exchange in opportunistic networks.

iClouds is another architecture built for spontaneous user interaction, and collaborative data exchange built on ad hoc Wi-Fi technology. It provides four information scanning and distribution mechanisms via exchange of information lists. However, it lacks mechanism to establish advanced communication i.e. it provides no mechanism to establish communication after mobile devices have exchanged information lists.

We believe that in opportunistic networks a faster neighbor discovery mechanism should exist, which will help devices in quickly gathering and analyzing neighbor information. It is also important to choose a stable neighbor based on statistics gathered during neighbor discovery. Stability is based on proximity and relative mobility of a neighbor. This will ensure that neighbor doesn’t go out of communication range when communication is in progress. Also, neighbor’s credibility should be lowered if it frequently leaves the neighborhood when communication is in progress.

After selecting the best available neighbor, next step is to begin communicating with this neighbor in synchronous request-response fashion. During this time, communication may break because of message loss or neighbor going out of range. If neighbor does go out of communication range, communication state should be saved for future reconnection. In future, if neighbor tries to reestablish communication, saved communication state can be restored and an attempt is made to start from the point the communication was broken.
4.2 Design Consideration

As neighbor discovery becomes an important constraint for communication in opportunistic networks, it is highly desirable to choose a short range communication technology that helps in finding more neighbors in fast and energy efficient manner. Two commonly available device discovery technologies are Bluetooth and ad hoc Wi-Fi. Bluetooth device discovery takes 10 to 17 seconds to complete, with an average of 12 seconds. [23] On a typical smartphone device, which is a class II Bluetooth device communication range is around 10 meters. [8] Wi-Fi protocol performs device discovery in 1-2 seconds. Also, typical Wi-Fi communication range on smartphones is 110 meters. [8] IEEE 802.11g standard enables maximum data rates up to 54 Mbps, as against to 3 Mbps data rate provided by Bluetooth 2.1 standard. Typical Bluetooth power consumption for transmission and reception is respectively 170 and 100 m Watts. Corresponding transmission and reception power consumption values for Wi-Fi protocol are 940 and 480 m Watts. [8] Thus while performing device discovery, Wi-Fi protocol is superior in terms of time and communication range, but Wi-Fi also has some energy overhead as compared to Bluetooth.

Consider a scenario where neighbors are distributed around the radius of 100 meters around our mobile device and are relatively immobile. Also, all devices have their Wi-Fi and Bluetooth connection ON, so that all of them can be identified using either Bluetooth or Wi-Fi device discovery. We can take Ad hoc Wi-Fi range for the device to be 100 meters and Bluetooth range to be 10 meters. Now, if we are standing at the center of this network, we will need at most one Wi-Fi device discovery query to discover almost all the neighbors. If we are standing at the edge of the network, we can still discover half the devices with one Wi-Fi device discovery query. Now, let us consider Bluetooth device discovery. If we are standing anywhere inside the network, only neighbors that are within 10 meter range of mobile device, will be discovered. To discover all the neighbors, observing node has to be mobile and should be able to cover entire region covered by Wi-Fi range in one scan, as shown in figure below. All
neighbors inside circular region will be discovered by respective device discovery query. For each distinct sub-region of Wi-Fi range, there has to be new Bluetooth device discovery query. If neighbors are assumed to be distributed uniformly, we require \((100)^2 / (10)^2 \approx 100\) Bluetooth device discovery queries to discover all distinct neighbors around. Same task can be performed using single Wi-Fi device discovery query. Effectively, to perform at par with Wi-Fi, Bluetooth protocol will have to expend considerably more power. Also, since opportunistic networks rely on user mobility, we cannot expect the observing node to cover all sub-regions. It should also be noted that Bluetooth device discovery is considerably slower as compared to Wi-Fi. This will affect device discovery performance when neighbors are mobile.

![Figure 4.1: Bluetooth Vs Wi-Fi Comparison](image)

As nodes in opportunistic networks are loosely connected and rely on user mobility, for reliable communication it is important that two nodes stay in each others’ range throughout communication session. With Wi-Fi, communicating devices can be expected to be mobile in the range of 100 meters, as against the 10 meter range for Bluetooth. This allows higher degree of mobility for communicating devices, which implies higher probability of session completion for Wi-Fi as compared to Bluetooth. Considering all above reasons and tradeoffs, we have decided to develop our application architecture based on ad hoc Wi-Fi technology.
4.3 Design and Implementation

We have designed two modules, each for resolving different issues in opportunistic networks. Module OppCommNetworking handles neighbor discovery issues, scans neighborhood periodically and provides pool of best neighbors to requesting applications. This pool of best neighbors is chosen on the basis of relative mobility and proximity of the neighbors.

OppCommFramework module optimizes the MultiCommFramework to suit the needs of opportunistic networking communication. This module sends application layer messages and listens to replies from neighbors, on behalf of the application above.

We have built two applications – an autonomous game of prisoner’s dilemma and a device discovery application. Block diagram for running opportunistic networking applications on our architecture is as follows:

![Application Architecture Diagram](image)

Figure 4.2: Application Architecture
4.3.1 OppCommNetworking

*OppCommNetworking* module handles neighbor discovery operations on behalf of application. This module also controls the wireless LAN card on mobile device and radio operations taking place on it.

4.3.1.1 OppCommNetworking Functions

In broad sense, OppCommNetworking module provides following functionalities:

- **Register device with framework:** When *OppCommNetworking* module is initialized, it publishes device’s 16 byte long identifier using beacon stuffing technique, discussed in previous chapter. We use SSID concatenation technique to publish device’s identifier, which uses 16 bytes of the 32 byte-capacity SSID field in beacon packets. After device is registered with framework, every beacon packet sends out device identifier information as a part of SSID field.

- **Unregister device from framework:** This operation disassociates device’s identifier information from IEEE 802.11 beacon packets. This operation is performed before closing down the framework.

- **Register application with framework:** This operation binds application identifier information with 802.11 beacon packets. After an application is registered, 802.11 beacon packets broadcast both application identifier and device identifier. This enables neighbors to discover application and device information on querying their wireless cards. Typically application identifier field is 8 bytes. While adding its information to 802.11 beacon, device and application identifier are separated by field separator.

- **Unregister application from framework:** This operation unbinds application identifier information from beacon packets. After performing this operation, beacon packets will only contain device identifier information and neighbors won’t be able to see the application running on device.
- Perform neighbor discovery: A background thread is started when OppCommNetworking module is initiated. This thread queries the wireless LAN card at fixed interval called ‘neighbor discovery interval’. The card returns list of nearby SSIDs. From the list of SSIDs, devices and applications running on them are identified. For every neighbor device identifier, list of applications, RSSI values are stored. We use NeighborRecord data structure to store information about every neighbor. Following is the structure of NeighborRecord.

<table>
<thead>
<tr>
<th>Device Identifier (16 bytes)</th>
<th>Field Separator (1 byte)</th>
<th>Application Identifier (8 bytes)</th>
</tr>
</thead>
</table>

Figure 4.3: Stuffed SSID field of IEEE 802.11 beacon

Time of contact value for a neighbor is calculated by multiplying the value ‘neighbor discovery interval’ and number of times neighbor was discovered continuously since its latest discovery i.e. if neighbor discovery interval is 10

Figure 4.4: NeighborRecord data structure
seconds and neighbor was discovered consecutively for 5 times including current query, then time of contact value is $5 \times 10 = 50$ seconds.

*NeighborRecords*, List of *NeighborRecord* objects is used by neighbor discovery thread to stores details about all neighbors. Since *NeighborRecords* is accessed simultaneously by multiple threads, we provide synchronous access to this data structure.

- Return available neighbors: When an application wants to find neighbors running similar application, it calls this operation. This operation returns the pool of neighbors running same application in the neighborhood. Different criteria can be used while selecting this pool.

1. **Maximum Time of Contact**: This approach chooses neighbors based on whether they cross a threshold of time of contact value. This approach focuses on stability of the neighbor i.e. a neighbor is chosen if it has been in the neighborhood for longer time.

2. **Maximum Average RSSI value**: This approach chooses a neighbor if the average of its RSSI values from previous scans crosses certain threshold. When a neighbor is not detected, its RSSI value is assumed to be minimum possible value (-95 dB). This approach focuses on proximity of a neighbor, and gives more weightage to neighbors that are nearer.

3. **Standard Deviation of RSSI values**: This approach chooses a neighbor if the standard deviation of its RSSI values from previous scans is below certain threshold. This approach focuses on relative mobility of the neighbor. If neighbor has been relatively still, more weightage is given to it.

4. **Maximum Time of Contact and Minimum RSSI Standard Deviation**: A neighbor is chosen based on Maximum Time of Contact approach. If multiple neighbors are found to have same time of contact value, the one
with minimum standard deviation of RSSI values is chosen. This approach favors relatively immobile neighbors among stable ones i.e. If there are numerous neighbors that have been in range for quite long, neighbors showing relative immobility get higher priority.

5. Maximum Time of Contact and Minimum RSSI Average: A neighbor is chosen based on Maximum Time of Contact approach. If multiple neighbors are found to have same time of contact value, the one with maximum average of RSSI values is chosen. This approach favors neighbor proximity among stable ones i.e. if there are numerous neighbors that have been in range for quite long, neighbors that are nearer get higher priority.

6. Connect to a neighbor: Apart from neighbor discovery operations, OppCommNetworking also performs connection establishment operation on behalf of application. When application requests to establish connection with a particular neighbor, this operation is called. Typically this operation takes around 10-15 seconds to complete, if successful. Following flowcharts shows how connection establishment is performed from requesting and responding node’s side.
Figure 4.5: Flowchart for the node sending Session Initiation Request
As shown in figure 4.5 and figure 4.6, nodes broadcast UDP invitation and invitation request messages, which is similar to MultiCommFramework. The main difference here is number of invitation messages are limited and their purpose is to get IP address and port...
number of the neighbor, not neighbor discovery as in *MultiCommFramework*. This IP address is required for sending unicast messages to neighbor once session starts.

When connection is established, communicating nodes are already on a unique SSID network, known as Session SSID network. An advantage of this approach is message exchange taking place on other SSID networks will be ignored at MAC layer. Also, having unique Session SSID network makes it possible for other nodes to realize that two nodes are interacting on a to recognize that there is an ongoing session communication in vicinity, also it makes it possible for neighbors to discard the busy or communicating devices.

4.3.1.2 Session Initiation Request message

Session Initiation Request message mentioned in figure 4.5 looks as follows:

<table>
<thead>
<tr>
<th>Session Request</th>
<th>DIS</th>
<th>Device ID</th>
<th>DIS</th>
<th>Application ID</th>
<th>DIS</th>
<th>Session SSID</th>
<th>DIS</th>
<th>Authentication Info (Optional)</th>
</tr>
</thead>
</table>

Figure 4.7: Session Initiation Request Message Format

'Session Request' field contains the request code for Session Initiation Request message. DIS stands for Data Item Separator. Requesting node sends device and application ID, so that neighbor's *OppCommNetworking* module can appropriate application callback function. Communicating nodes can optionally choose to communicate over secure channel. Various authentication modes available are: WEP 64 or 128 bit, WPA (TKIP) and No WPA2 (AES). [8] Information regarding authentication can be stored in ‘Authentication Info’ field. Currently, we use Open System authentication with data encryption disabled, to avoid communication overhead and time spent in analyzing security parameters.
4.3.1.3 **Session SSID**

A typical Session SSID looks as follows. Sizes mentioned are in bytes. Separator is a character helps in identifying the presence of session communication in vicinity. App ID denotes the application running on communicating parties. It is different from the 16 byte Application ID used in communication messages. Part of device ID fields stores only half of the device ID i.e. 8 bytes.

<table>
<thead>
<tr>
<th>Separator (1)</th>
<th>App ID (8)</th>
<th>Separator (1)</th>
<th>Part of device ID (8)</th>
<th>Separator (1)</th>
<th>Part of device ID (8)</th>
</tr>
</thead>
</table>

Figure 4.8: Session SSID - Stuffed SSID field of IEEE 802.11 beacon

4.3.1.4 **Wrapper Libraries**

*OppCommNetworking* also provides wrapper libraries for Smart Device Framework to control wireless LAN card and Wi-Fi/Bluetooth radio operations. Its main aim is to provide simple interface to applications and allow them to focus only on application layer issues. These wrappers expose singleton instances of Wireless LAN card and radio controller. Following class diagrams give brief overview of their functionalities.

![RadioController Diagram](image)

Figure 4.9: RadioController – Wrapper to control Bluetooth and Wi-Fi radio operations
4.3.2 OppCommFramework

OppCommFramework module is broadly based on MultiCommFramework architecture developed by Jim Wilson. [20] As MultiCommFramework is based on the model of multicasting, hence not ideal for opportunistic networks as opportunistic networks rely on one-to-one communication. So, we limit the number of clients in OppCommFramework to one.

Also, device discovery in MultiCommFramework is performed based on application layer invitations as UDP broadcast messages. This scheme requires clients and listener devices to broadcast invitations until they are replied by other party. To gather information from multiple devices, a device has to receive different invitation messages or five different replies. Instead, if a device queries the underlying wireless card to get the list of neighbors, all replies are received
within a span of 1-2 seconds. Since device discovery is performed at MAC layer, time spent in sending application layer message to MAC layer and receiving reply from MAC layer is saved.

In our architecture, neighbor discovery and connection establishment part is handled by OppCommNetworking module. Once two neighbors are connected to each other i.e. once they are associated with same Session SSID, control is handed over to OppCommFramework module. OppCommFramework still uses UDP broadcast messages like MultiCommFramework; however purpose of these messages is to get the IP address of neighbor, so that UDP/IP communication can be initiated. Also, when two nodes are communicating over a unique Session SSID network, interference from network communication on other SSID networks will be avoided, as those messages will not come above MAC layer.

OppCommFramework also accommodates information about application identifier in communication messages. This enables multiple applications to be built on top of OppCommFramework. Although, current architecture allows only one application to run at a time, OppCommFramework easily accommodates different applications running at different times.

A typical communication message based on OppCommFramework has following format:

<table>
<thead>
<tr>
<th>Device ID</th>
<th>Application ID</th>
<th>Sequence #</th>
<th>OpCode</th>
<th>Message Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>(16 bytes)</td>
<td>(16 bytes)</td>
<td>(4 bytes)</td>
<td>(1 byte)</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.11: Communication message format in OppCommFramework

Device ID is same as the one used during beacon stuffing mechanism. Application ID is a unique identifier for every application. Sequence # keeps track of the current message received, and discards duplicate or out-of-sequence messages. OpCode is same as the one used during MultiCommFramework communication. Chapter 3.2 – MultiCommFramework
describes different types of OpCodes in detail. Remaining bytes store the data related respective OpCodes.

4.3.3 Opportunistic Networking Application: Iterated Prisoner’s Dilemma

The application of prisoner’s dilemma is built around the game of iterated prisoner’s dilemma. In single round prisoner’s dilemma game, two players act as prisoners who can choose to cooperate with each other or defect against each other. While playing, the players do not know about each other’s moves. Each player gets points based on following table.

<table>
<thead>
<tr>
<th></th>
<th>Cooperate (Player 2)</th>
<th>Defect (Player 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cooperate (Player 1)</td>
<td>3,3</td>
<td>0,5</td>
</tr>
<tr>
<td>Defect (Player 1)</td>
<td>5,0</td>
<td>1,1</td>
</tr>
</tbody>
</table>

In iterated prisoner’s dilemma, both players know about previous moves made by other players. So, player chooses the strategy for current round based on historical information. For example, a player might choose from following strategies:

1. Tit-For-Tat: If opponent defected during last round, the play ‘Defect’, otherwise play ‘Cooperate’.
2. Grudger: If opponent defected even once, always play ‘Defect’, thus holding grudge forever.
3. Random: Play each move irrespective of what move was made by opponent.

When players are playing these games in an opportunistic network, two players are represented by two mobile devices communicating with each other. Player chooses a strategy in the beginning of game and always communicates last played move with its opponent. Current
move is chosen based on previous move received from opponent. During this communication, following issues may arise.

1. Communication synchronicity: When communication is in progress, it is important to receive only previous round’s move from opponent i.e. when two players are playing for round two, first player will send its round one move and second player will reply with its round one move. Any other order of message sending and reception is undesired. Also, a player doesn’t play current move unless it has received previous move from its opponent. This makes the communication synchronous in nature. In order to preserve this synchronicity, our application keeps track of currently expected move, and any other or duplicate moves are received, they are discarded.

2. Connection break and restoration: Since opportunistic networks are loosely connected, sometimes mobile devices may go out of range from each other when communication is in progress. Also, a communication message may get lost or might be received out of order. In all these cases, communication is assumed to be broken. As explained before, message communication is handled by \textit{OppCommFramework} module. When a communication session is started, session file is created that corresponds to Session SSID network, thus a session that is unique for two neighbors and corresponding roles they are playing during communication – host and client. When session is completed successfully, session file is deleted. If session doesn’t go to completion, application enters pause mode and session file is saved. Next time, when same neighbor comes into contact with current node with similar role, previous session file is used reestablish broken connection. Following diagram explains it in detail.
3. Neighbor credibility: In opportunistic networks, a node can come into contact with many unknown nodes. Thus before initializing communication, it is important to know how credible the neighbor is i.e. whether communication will go to completion or not. If communication with an opponent frequently breaks down, it might not be an intelligent choice to communicate with that neighbor, next time when it is discovered. This gives rise to the notion of credibility and it is defined as follows:
\[ \text{Credibility} = \frac{\text{number of completed sessions}}{\text{number of total sessions}} \]

Application stores credibility information about every discovered neighbor in ‘credibility data store’.

4.3.4 Development and Testing Environment

The mobile device used for testing is HP iPAQ 910 Business Messenger. The device has Marvell PXA 270 processor operating at 426 MHZ and it runs on Windows Mobile 6.1 Professional operating system. It comes integrated with IEEE 802.11b/g with WPA2 security and Bluetooth 2.0 with EDR support. It has 128 MB SDRAM main memory and 256 MB ROM memory, Lithium Ion battery (1940 mAh) and 320X240 pixel touch panel display with 65000 colors.

We develop our applications for Windows Mobile 6 platform. We use .NET compact framework 2.0 from Microsoft and third party library - Smart Device Framework 2.3 developed by OpenNETCF Consulting. This library provides basic functionality to interact with wireless LAN card and control radio operations on Window Mobile devices.

Development, deployment and testing tools are as follows:

1. Microsoft Visual Studio 2005
2. Windows Mobile 6 Software Development Kit
3. Microsoft ActiveSync: For synchronizing files on desktop and mobile device.
CHAPTER 5
EXPERIMENTS AND RESULTS

In our experiments, we focus on experiments analyzing the performance of various neighbor discovery techniques. We have developed device discovery application as a data collection application that runs on top of developed architecture. (See figure 4.2)

5.1 Opportunistic Networking Application: Device Discovery

This application lets user enter following device discovery parameters:

1. Device discovery interval (tTimeOut)
2. Minimum sleep time during consecutive discoveries (tSleepMin)
3. Maximum sleep time during consecutive discoveries (tSleepMax)

Currently only device discovery interval is utilized to gather neighbor data at constant intervals. As discussed before, all neighbors use beacon stuffing technique to publish the device and application information. A typical SSID field in IEEE 802.11 beacons can be seen in figure 4.3: Stuffed SSID field of IEEE 802.11 beacon. Data collection is performed as follows:

1. User selects device discovery interval in seconds.
2. Application initiates OppCommNetworking module and registers application with it.
3. Application initiates timer thread that queries wireless LAN card after every tTimeOut seconds.
4. On querying, wireless LAN card returns the list of nearby ad hoc access points, which contains SSID, signal strength, signal quality, channel number.
5. Application stores date and time of scan, SSID information, RSSI value and RSSI quality of the signal received for every discovered neighbor, in a csv file.

6. Application continues to run until user chooses to stop it or battery level drops below critical level.

5.2 Device Discovery Comparison

Bluetooth device discovery typically takes anything from 10 to 17 seconds of time, with an average of 12 seconds. Also, Bluetooth device discovery is effective in finding neighbors in the range of 9-12 meters. Bluetooth device discovery time varies based on number of devices in the neighborhood. [23]

Wi-Fi device discovery mechanism takes around 1-2 seconds. It is effective in finding neighbors in the range of 100-110 meters. The discovery time is independent of the number of nodes in the neighborhood. This implies that Wi-Fi has clear advantage over Bluetooth technology, when it comes to device discovery.

5.3 Neighbor Selection Performance Evaluation

After neighbor data is collected from every neighbor, time of contact data for every neighbor is found out. Time of contact (TOC) for a neighbor is calculated from device discovery interval (tTimeOut) and number of consecutive scans during which this neighbor was found (n), as follows:

\[
TOC = t\text{TimeOut} \times n
\]

Time of contact is used in following experiments to analyze the performance of various neighbor selection strategies:
1. Maximum Time of Contact: At given instance, a neighbor with maximum time of contact is chosen for communication. This approach favors stability of neighbor with respect to node i.e. if node that has been in neighborhood for long time is preferred for communication.

2. Minimum Time of Contact: At given instance, a neighbor with minimum time of contact is chosen for communication. The approach favors newer neighbors over older ones.

3. Standard deviation of RSSI values: At given instance, a neighbor with minimum standard deviation of its RSSI values is chosen. This approach favors relative immobility of a neighbor i.e. node that has been at approximately same distance for long time is preferred for communication.

4. Average of RSSI values: At given instance, a neighbor with maximum average of RSSI values is chosen. This approach favors the proximity of neighbor i.e. a node that is nearest is chosen for communication.

5. Time of Contact and standard deviation of RSSI values: At given instance, a neighbor is chosen for communication if it has maximum time of contact with given node. If multiple neighbors have same time of contact, then preference is given to the neighbor with minimum standard deviation of RSSI values. This approach chooses nodes with maximum time of contact and then favors relatively immobile nodes over the mobile ones.

6. Time of Contact and average of RSSI values: At given instance, a neighbor is chosen for communication if it has maximum time of contact with given node. If multiple nodes have same time of contact, then preference is given to the neighbor with maximum average of RSSI values. This approach chooses nodes with maximum time of contact and then favors nearer nodes over farther ones.
At any given instance, neighbors chosen by all above strategies is compared against optimal neighbor. Optimal neighbor is the one that will be in contact with given node for longest time and is sufficiently near to it (Based on RSSI value threshold of -82 dB).

5.3.1 **Experiment Setup**

We performed experiments in different environments – inside shopping mall, inside lab environment and outside environment. Every mobile device involved in experiment runs Device Discovery application mentioned above and broadcasts SSID information about itself via beacon stuffing. Device Discovery interval is set to be 5 to 10 seconds. Each node queries wireless LAN card to scan the neighborhood, and stores information about date, time, SSID, RSSI value and RSSI quality information. Based on this information, we first calculate time of contact distribution. Time of contact distribution maps different session intervals with their occurrences. Then we find out optimal neighbor at every instance as described in previous paragraph. This optimal neighbor is compared against neighbor calculated by every strategy. Whenever there is a match, that strategy is said to have performed better at given instance. Similar calculations are performed for every instance to get percentage accuracy of every strategy in given environment. Next, we discuss our test results in different environments.

5.3.1.1 **Open Space**

In this environment, users were carrying mobile devices and moving in open space independently. Following is the time of contact distribution, where time of contact varies from 0 to 4160 seconds and occurrences vary from almost 380 to 0.
There is an unusual peak at around 40 second, showing that 40 second long sessions occurred more often than 30 or 50 second sessions.
Figure 5.2: Performance Analysis of Neighbor Selection Algorithms (Open Space)

Above figure shows the performance of neighbor selection strategies. It shows that neighbors chosen based on maximum time of contact, maximum RSSI average and minimum RSSI standard deviation, give much better estimate as against minimum time of contact approach. This implies that nodes that have just arrived in the neighborhood may not be a good choice for communication.

We also compare these approaches against randomly chosen neighbor and find that all approaches, except minimum time of contact approach yields better results. Moreover, combination of two approaches gives still better estimate while selecting best neighbor. Neighbors chosen using time of contact and minimum RSSI standard deviation, time of contact and maximum RSSI average perform significantly better than any single approach.
5.3.1.2 Shopping Mall Environment

In this environment, users were given mobile devices and they were moving around in shopping mall independently. Following is the time of contact distribution.

![Time of Contact Distribution](image)

Figure 5.3: Time of Contact Distribution (Shopping Mall)

The above graph shows some unusually high peaks at around 60-70 seconds time of contact period. This peaks occur later as compared to the peaks appeared in figure 5.1 i.e. more sessions occur more often in this environment, as compared to the previous one. This might be attributed to closed environment of shopping mall, restricting user movement outside each other’s communication boundaries, thus staying in contact for longer time.
Figure 5.4: Performance Analysis of Neighbor Selection Algorithms (Shopping Mall)

We observe that although percentage performance of neighbor selection strategies is reduced, a combination of two approaches still performs slightly better than most of the single approaches. RSSI average and standard deviation strategies don’t perform as good as maximum time of contact strategy. This might be because closed environment causes more interference with RSSI values.

5.3.1.3 Closed Lab Environment

Further we perform our experiments inside closed lab space. Users are carrying mobile devices and moving around in lab independently. Figure below shows the time of contact distribution for this experiment.
Above figure shows slightly high peaks at around 70 seconds time of contact interval. As compared to experiments conducted in open space, time of contact here is longer. This could again be attributed to closed lab environment forcing users to stay in lab for longer time.
Figure 5.6: Performance Analysis of Neighbor Selection Algorithms (Closed Lab Environment)

We observe slight decrease in performances of all the strategies, still combination of approaches outperform all single approaches. Again, all approaches except minimum time of contact perform better than random neighbor approach.

5.4 Credibility Tests

Credibility tests are performed to analyze the effect of credibility on the performance of communication success. As mentioned before, credibility is calculated as

\[ \text{Credibility} = \frac{\text{number of completed sessions}}{\text{number of total sessions}} \]
We consider two scenarios, one in which nodes select neighbors based on credibility and other in which neighbors select neighbors randomly. We conduct experiments over the span of 5-6 hours that ensure around 950 contacts in either case.

![Bar chart showing contacts with and without credibility](image)

Figure 5.7: Credibility tests results

From above results, we see that there is almost four times increase in number of successful contacts upon using credibility before selecting neighbors. Large number of unsuccessful contacts is attributed to high level of node mobility in opportunistic networks and to unpredictable wireless LAN card behavior.

5.5 Network Statistics

We calculated average network delays and average time spent on a session SSID network by nodes in opportunistic network.

- Average Time to switch to session SSID = 14.63 seconds
• Average Time spent on session SSID for successful contact (10 rounds)= 105 seconds

• Average Time spent on session SSID for successful contact (for 20 rounds)= 144 seconds

• Average Time spent on session SSID for unsuccessful contact= 70 seconds
CHAPTER 6
CONCLUSION AND FUTURE WORK

Today’s exploding mobile phone industry and advanced capabilities of mobile phones have generated huge interest in development of smartphone applications that can interact with each other. Opportunistic networks are one such emerging class of smartphone applications. These applications can exchange information with neighboring devices without much user intervention. They exploit user proximity and mobility to share and disseminate information.

Thus it is hard to predict the group formation or communication pattern in opportunistic networks. This gives rise to numerous communication issues unique to opportunistic networks. These issues can be broadly categorized as neighbor discovery and one hop message exchange between neighboring devices.

We have designed and implemented an application architecture that addresses the needs of opportunistic networks. Currently there are numerous solutions available based on Bluetooth short range communication technology. Few existing solutions are based on ad hoc Wi-Fi technology satisfy partial needs of opportunistic networks. Ad hoc Wi-Fi protocol has many advantages over Bluetooth, in terms of faster device discovery, greater opportunistic contact period and more contact opportunities. We embed application and device information in MAC layer beacon frames, thus providing speedy and energy efficient solution for neighbor discovery. We then compare the device discovery performance of typical Bluetooth application with our device discovery mechanism and find it to be significantly better.

In opportunistic networks, at any given time there can be more than one useful neighbor in the surroundings. In case of interactive applications, it is important to choose the neighbor that will take interaction to completion. Hence it is vital to decide on how to choose a neighbor.
We provide several neighbor selection mechanisms based on history of time of contact and previous signal strength values. We analyze the performance of these mechanisms by conducting series of experiments in different environments. We prove that using neighbor selection mechanisms can help nodes select good neighbors so that interaction goes to completion. At the end of every successful or unsuccessful interaction, we update neighbor’s credibility factor for future encounters.

Furthermore, we enhance an existing multicast communication framework to suit the needs of opportunistic networks. We add support for building multiple applications and limit communication to only one hop message exchange between two nodes. In our architecture, messages are exchanged on UDP/IP protocol and also, we support synchronous message exchange between devices.

We implemented our architecture on HP iPAQ 910 devices and built prisoner’s dilemma game as a demo application running on this architecture. We realized that having dedicated application architecture makes application development easy, and helps developers focus more on application level issues. Also, we built device discovery application to gather data about neighbors. This application provides interface for configuring device discovery mechanism inside architecture.

In short, our application architecture handles neighbor discovery and message exchange operations for applications built on opportunistic networks. We would like to further our research by improving performance of connection re-establishment mechanism provided inside architecture. We would also like to conduct experiments with more mobile devices in neighborhood. This will give us better picture of application layer performance in terms of network delays, network utilization, effect of network density, and battery usage.
REFERENCES


Sankalp Vinod Shere was born July 1985, India. He received his Bachelor of Engineering in Computer Engineering from VJTI, University of Mumbai, India in June 2007.

In fall of 2007, he started his graduate studies in computer engineering. In summer of 2008, he did application development internship at Ayoka, Arlington TX. In spring 2009, he did software engineering internship at Siemens Energy & Automation, Irving, TX. He received his Masters in Computer Engineering from the University of Texas at Arlington, in Dec 2009. His research interests include wireless sensor networks, opportunistic networks and mobile social networking.