2014

Towards Secure, Power-Efficient and Location-Aware Mobile Computing

Hao Han
College of William & Mary - Arts & Sciences

Towards Secure, Power-Efficient and Location-Aware Mobile Computing

Hao Han

Shanghai, China

Bachelor of Science, Nanjing University, 2005

A Dissertation presented to the Graduate Faculty
of the College of William and Mary in Candidacy for the Degree of
Doctor of Philosophy

Department of Computer Science

The College of William and Mary
January, 2014

APPROVAL PAGE

This Dissertation is submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

[illegible]

Hao Han

Approved by the Committee, January 2014

[illegible]

Committee Chair
Associate Professor Qun Li, Computer Science
The College of William and Mary

Weizhen Mao

Professor Weizhen Mao, Computer Science
The College of William and Mary

[illegible]

Professor Evgenia Smirni, Computer Science
The College of William and Mary

[illegible]

Associate Professor Gang Zhou, Computer Science
The College of William and Mary

[illegible]

Dr. Emiliano Miluzzo
AT&T Labs - Research

ABSTRACT

In the post-PC era, mobile devices will replace desktops and become the main personal computer for many people. People rely on mobile devices such as smartphones and tablets for everything in their daily lives. A common requirement for mobile computing is wireless communication. It allows mobile devices to fetch remote resources easily. Unfortunately, the increasing demand of the mobility brings many new wireless management challenges such as security, energy-saving and location-awareness. These challenges have already impeded the advancement of mobile systems. In this dissertation we attempt to discover the guidelines of how to mitigate these problems through three general communication patterns in 802.11 wireless networks. We propose a cross-section of a few interesting and important enhancements to manage wireless connectivity. These enhancements provide useful primitives for the design of next-generation mobile systems in the future.

Specifically, we improve the association mechanism for wireless clients to defend against rogue wireless Access Points (APs) in Wireless LANs (WLANs) and vehicular networks. Real-world prototype systems confirm that our scheme can achieve high accuracy to detect even sophisticated rogue APs under various network conditions. We also develop a power-efficient system to reduce the energy consumption for mobile devices working as software-defined APs. Experimental results show that our system allows the Wi-Fi interface to sleep for up to 88% of the total time in several different applications and reduce the system energy by up to 33%. We achieve this while retaining comparable user experiences. Finally, we design a fine-grained scalable group localization algorithm to enable location-aware wireless communication. Our prototype implemented on commercial smartphones proves that our algorithm can quickly locate a group of mobile devices with centimeter-level accuracy.

TABLE OF CONTENTS

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to many:

To my advisor, Dr. Qun Li, who has been an endless source of patience, knowledge, and encouragement. This dissertation could not have been done without his thoughtful guidance. I am greatly indebted to his commitment of time and energy, constructive critique and generous support in every step of my research. I could not have imagined having a better advisor for my Ph.D. study.

To my committee members, Dr. Weizhen Mao, Dr. Evgenia Smirni, Dr. Gang Zhou, Dr. Emiliano Miluzzo for generosity of time and support, as well as insightful comments.

To many others in Computer Science Department at the College of William and Mary, especially Vanessa Godwin, Jacqulyn Johnson, and Dale Hayes for their kindly helps.

To Dr. Yunxin Liu, Dr. Guobin Shen, and Dr. Yongguang Zhang at Microsoft Research Asia for offering me the internship opportunity in their groups and leading me working on diverse exciting projects.

To my classmates, labmates, and co-authors, especially Dr. Fengyuan Xu, Dr. Wei Wei, Yifan Zhang, Zhenrui Qin, Ed Novak, Dr. Chiu C. Tan, Dr. Bo sheng, and Dr. Haodong Wang and many other dear friends for their company and support.

To my wife, Shuai Yang, for her unconditional love, faith, and sincerity. To her family, for welcoming me with warmth, respect, and without reservation. To my family, for supporting me throughout my life.

To my family

LIST OF TABLES

LIST OF FIGURES

1 Introduction

The world has become increasingly mobile. As a result, mobile devices such as smartphones and tablets, as the embodiment of today's mobile computing, have replaced desktops to become the main personal computing platforms. According to Cisco's prediction, by the end of 2013, the number of mobile-connected devices will exceed the number of people on this planet, and by 2017 there will be nearly 1.4 mobile devices per capita [5]. Not only has there been extraordinary growth in popularity, the diversity of mobile devices has also increased rapidly. New types of mobile devices such as "smart" watches, glasses and even driverless vehicles have emerged recently. In our homes and enterprises, our recreation and profession, mobile computing is playing a more and more important role. It is becoming an essential and ubiquitous tool of the modern world.

Communication is the core of mobile computing. It allows mobile devices to connect to networks and fetch remote resources. Due to the increasing demand of mobility, traditional wired networks are now insufficient. If users have to use cables for communication, their movement will be significantly reduced. Wireless communication has no such restriction and thereby becomes an indispensable feature to mobile computing. The majority of today's mobile devices have already integrated multiple interfaces to support various wireless networks such as 802.11 networks (a.k.a., Wi-Fi), cellular networks and Bluetooth networks.

In the past decade, wireless communication has been greatly improved in bandwidth and latency. However, mobility brings new challenges such as security, energy-efficiency and location-awareness. Security is of growing importance as more and more personal data are stored on mobile devices and transmitted through wireless communications. Without careful protection, users' privacy will be violated. Unlike workstations behind the firewalls, mobile devices typically operate under risky circumstances, so they have more opportunities to be attacked by adversaries. To improve mobile security, mobile devices must have the capability to defend against various attacks on their own. Energy is another important consideration for mobile devices. Mobility prevents mobile devices from always being plugged in to power sources. Insufficient battery life has seriously impeded the advancement of mobile technology. Given the fact that today's battery technology has hardly improved, while wireless networking is a significant contributor to battery drain, more efficient power-saving techniques for wireless communication are highly desired. Finally, location-aware wireless communication becomes useful due to mobility. Knowing the location allows mobile devices to adapt to varying environments and thereby improve the quality of wireless communication. Location-based techniques like this could also bridge the perception gap between human and machine, thus enabling more intuitive methods of communication such as transferring data by swinging a device towards the target user. Furthermore, location could help secure wireless communication (e.g., distance-bounding). However, there is little research on how to obtain the in-situ location of mobile devices without the support of network infrastructure. It is advantageous to design new localization schemes that can be applicable to anywhere. All the concerns rising from the security, energy and localization motivate our work presented in this dissertation.

Our research seeks to mitigate the problems of security, energy, and localization for mobile computing, but these problems exist in so many scenarios that we cannot enumerate all of them. In this dissertation we only consider a few enhancements

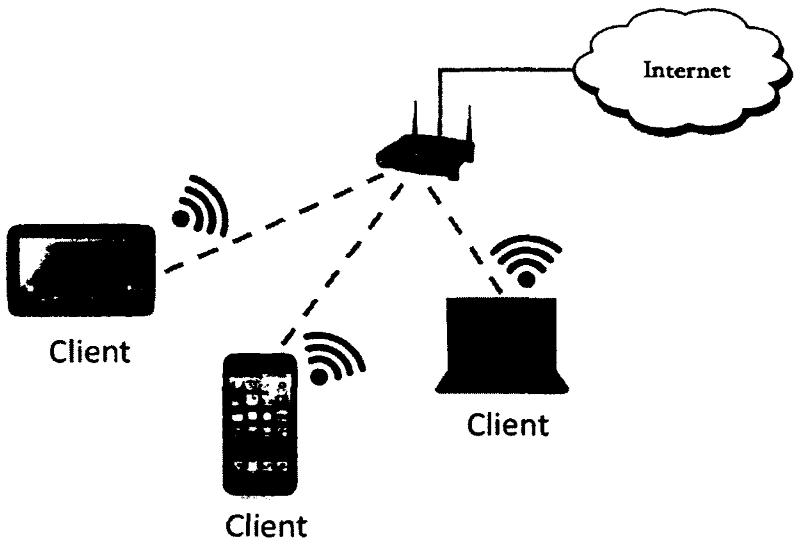


Figure 1.1: Illustration of mobile devices acting as clients

for managing wireless connectivity in 802.11 networks. These enhancements will shed light on a more generalized design of the next-generation mobile systems. In any 802.11 wireless network, a mobile device must operate in one of three modes: *client*, *AP* and *ad hoc*. Based on this division, 802.11 wireless communication can be broadly classified into three patterns: *client-to-AP*, *AP-to-client* and *client-to-client*. By studying individual problems for different communication patterns, we attempt to discover the guidelines for the security, energy management and localization enhancements.

For security, we study the most common communication pattern: client-to-AP, which is illustrated in Fig. 1.1. As an important part of this pattern, the wireless Access Point (AP) is a bridge that connects wireless clients to wired backbone networks. Mobile devices working as wireless clients associate to an AP to transfer data. It is worth noting that clients cannot directly communicate with each other. All the communications have to go through the AP which then routes the packets to the desired destination. Thus, a critical security issue arises for each client: *How to determine which AP is a legitimate AP for association?* A legitimate AP is referred to as the AP deployed by network administrators, whereas a rogue AP is

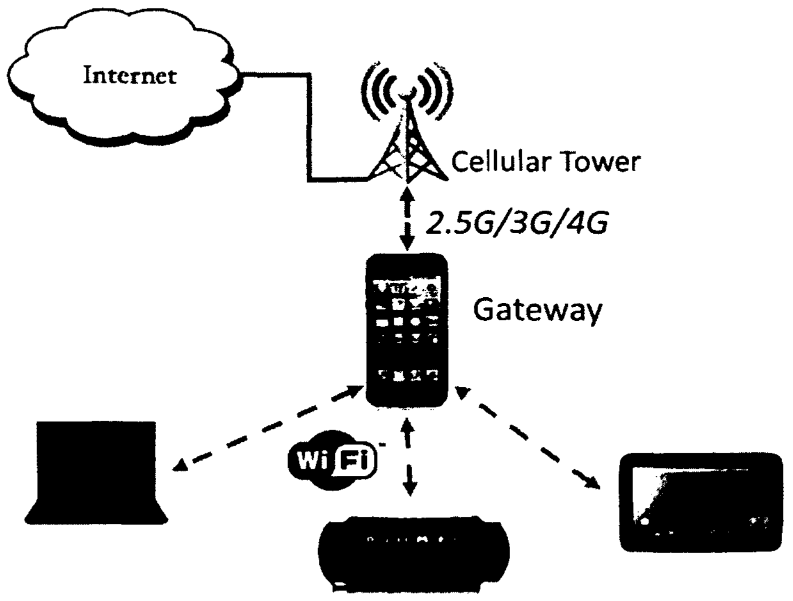


Figure 1.2: Illustration of a mobile device acting as a softAP

set up by the adversary who seeks to steal clients' information. A rogue AP will try every method to masquerade as a legitimate AP to avoid detection. Therefore, it is challenging for a client to detect rogue APs. We investigate this security problem and propose specific detection schemes for different network architectures.

As for energy issues, we study a new communication pattern for mobile devices. In this pattern, a mobile device performs as a software-defined AP, allowing other devices such as laptops and tablets to achieve everything from file transfer to Internet connectivity without relying on existing traditional APs. In the literature, these types of devices are typically called softAPs. Fig. 1.2 shows an example of a softAP sharing cellular connectivity such as 3G or 4G through Wi-Fi. Other devices can then associate to share the Internet access. Energy issues are rarely considered in this new pattern, because traditional APs do not rely on batteries. Old wisdom tells us that performance is more important than the energy. However, this is not true anymore for battery-powered mobile devices. Our initial findings can shed light for further research on this topic.

As mobile users are increasingly grouped to exchange files or perform collab-

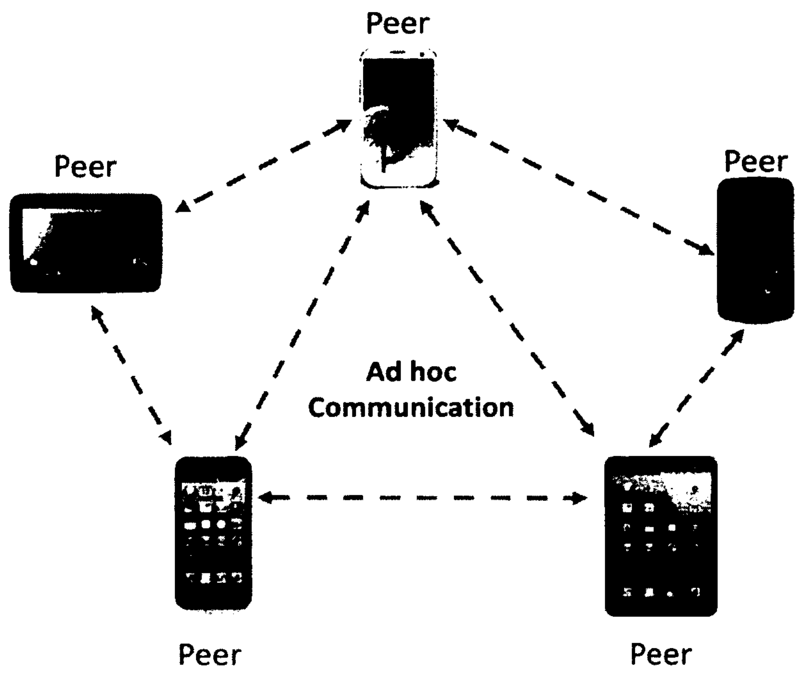


Figure 1.3: Illustration of mobile devices acting as peers in ad hoc wireless networks

orative tasks when meeting together, the ad hoc communication pattern becomes more and more popular. To improve this communication pattern, we strive to enable location-aware wireless networks. Fig. 1.3 shows a typical wireless ad hoc network. In this network, inter-connected mobile devices can directly communicate with each other. The ad hoc pattern increases the demand for more intuitive ways to identify communication parties. In traditional networks, devices are labeled with unique network addresses. To transfer data, the sender has to specify the network address of the destination. This is not intuitive for humans, because the addresses are long and semi-arbitrary. As a complimentary approach to the network address, we propose the use of location to establish wireless communication. For example, users can simply swing their device towards the destination devices to transfer files. This technique will bridge the perception gap between the physical world and digital world. The future development of mobile applications will benefit from our scheme.

In summary, the architecture of this dissertation is shown in Fig. 1.4. Across

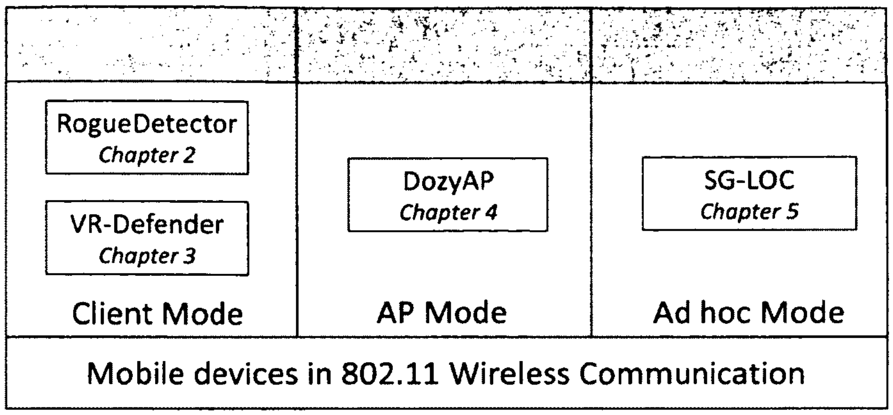


Figure 1.4: Architecture of the dissertation. Across three general communication patterns, a few useful and important enhancements are grouped by security, energy and location

three general patterns of wireless communications, where mobile devices operate in client, AP and ad hoc modes, we propose three useful and important enhancements to improve security, energy, and localization.

1.1 Problems

Specifically we address four particular problems as follows.

1.1.1 Rogue AP detection in Wi-Fi networks

Public areas such as coffee shops, airports, and campuses all provide Wi-Fi access to users, where people can use mobile devices such as smartphones and tablets to access the Internet freely. This thesis presents a practical timing-based technique to prevent wireless clients from connecting to potentially rogue APs. Our detection scheme is a pure user-centric approach that only leverages the Round Trip Time (RTT) between clients and local Domain Name System (DNS) resolvers without any support from the network administrators. We implemented the detection technique on commercially available mobile devices and evaluated the performance based on extensive experiments. Our detection scheme achieves more than 90% accuracy in distinguishing rogue APs from legitimate APs under lightweight traffic conditions and more than 60% accuracy under heavyweight traffic conditions.

1.1.2 Vehicular rogue AP detection

Increasingly, city-wide Wi-Fi networks are deployed around the world. As a result, people in vehicles can access the Internet via roadside APs, called *Drive-thru* Internet [7]. This thesis considers the problem of rogue APs that are set up in moving vehicles seeking to lure mobile users to associate by mimicking legitimate roadside APs. Because the users are moving in vehicles, and the non-malicious APs are roadside, and therefore immobile, users will leave the radio range of each AP and connect to another AP periodically. Vehicular rogue APs, however, are able to maintain a long connection with users, thus having more time to launch various attacks. We propose a practical user-side detection scheme to prevent users from connecting to vehicular rogue APs. Users can merely measure the received signal strength of beacons broadcasted by APs and actively send several messages to validate whether an AP is a rogue AP. We have implemented and evaluated our detection technique on commercial off-the-shelf devices in the real world. We observed that our detection scheme can achieve more than 90% accuracy in the experiments.

1.1.3 Power saving for Wi-Fi tethering

Wi-Fi tethering (i.e., sharing the Internet connection of a mobile phone via its Wi-Fi interface) is a useful feature and is widely supported on commercial smartphones. Yet existing Wi-Fi tethering schemes consume excessive power; they keep the Wi-Fi interface in a high power state regardless if there is ongoing traffic or not. In this thesis we propose DozyAP to improve the power efficiency of Wi-Fi tethering. Based on measurements in typical applications, we identify many opportunities for a tethering phone to sleep to save power. We design a simple yet reliable sleep protocol to coordinate the sleep schedule of the tethering phone with its clients without requiring tight time synchronization. Furthermore, we develop a two-stage, sleep interval adaptation algorithm to automatically adapt the sleep intervals to fit ongoing traffic patterns of various applications. DozyAP does not require any changes to the 802.11 protocol and is incrementally deployable through software updates. We have implemented DozyAP on commercial smartphones. Experimental results show that, while retaining comparable user experiences, our implementation can allow the Wi-Fi interface to sleep for up to 88% of the total time in several different applications, and reduce the system power consumption by up to 33% under the restricted programmability of current Wi-Fi hardware.

1.1.4 Group Localization for wireless networks

Mobile devices are often grouped to exchange files or perform collaborative tasks when meeting together. In these local groups, the relative locations of other group members is critical to enabling location-aware applications for wireless communication. Ideally, the localization process should be fine-grained, efficient, low-cost, scalable, and user-friendly. Existing solutions do not satisfy all of these criteria. They are some techniques that can only provide room-level positioning resolution, while other techniques require a strong collaboration between every device to exchange information or synchronize time. Some techniques rely on infrastructure support, and others require special hardware. In this thesis we consider these criteria and propose SG-LOC, a system of grouping and locating mobile phone users by performing a simple gesture. We have implemented and evaluated our system on commercial smartphones. Our experiments have shown that SG-LOC can achieve less than 1° orientation error and can successfully build a simple map of five people in an office room with 20cm error on average.

1.2 Contributions

This thesis contributes to the following three aspects: mobile security, energy management of wireless networks and mobile localization. Each aspect is described as follows.

1.2.1 Contributions to mobile security

The rogue AP problem is a critical security problem for wireless clients. However, there is little research on rogue AP detection, especially regarding "smart" but malicious rogue APs. In this thesis, we deeply investigate both static and mobile rogue APs that are likely to be set up in the real world. Our main contributions are as follows:

1. We are the first to propose a timing-based detection scheme to defend against static rogue AP in 802.11 networks. Our solution can detect sophisticated rogue APs that try to avoid detections as opposed to "accidental" rogue APs, for example, deployed by an innocent employee in an office. The detection algorithm can be purely performed by end users without any help from network administrators.

2. We are the first to study the mobile rogue AP problem in vehicular networks. Similarly, a novel user-side detection algorithm is designed to reduce the risk of "man-in-the-middle" attacks.

3. Our schemes are compatible with 802.11 standards. We have implemented prototype systems on commercial hardware and evaluated them through real-world experiments. Our experience and findings will help the further research on this topic.

1.2.2 Contributions to energy management

To the best of our knowledge, we are the first to study the important problem of how to save energy for softAPs in Wi-Fi tethering. The main contributions are in the following aspects:

- We study the characteristics of existing Wi-Fi tethering. We show that current Wi-Fi tethering is power hungry, wasting energy unnecessarily. We analyze the traffic patterns of various applications and identify many opportunities to optimize the power consumption of Wi-Fi tethering.
- We propose DozyAP to improve power efficiency of Wi-Fi tethering. We design a simple yet reliable sleep protocol to schedule a mobile softAP to sleep without requiring tight time synchronization between the softAP and its clients. We develop a two-stage adaptive sleep algorithm to allow a mobile softAP to automatically adapt to the traffic load for the best sleep schedule.
- We implement the DozyAP system on commercially available off-the-shelf (COTS) smartphones and evaluate its performance through experiments with real applications and simulations based on real user traces. Our evaluation results show that DozyAP is able to significantly reduce power consumption of Wi-Fi tethering and retain comparable user experience at the same time.

1.2.3 Contributions to mobile localization

This is the first work integrating commercial hardware such as Inertial Measurement Units (IMUs), speakers and microphones to locate mobile devices without the support of network infrastructure. The main contributions are as follows:

- We develop SG-LOC, a system to efficiently group and locate mobile users in proximity, where the localization process cleverly leverages dead reckoning and multilateration algorithms without any infrastructure support.

- We characterize the challenges of implementing SG-LOC, namely significant drift in integration of sensors, signal detection in environment with highly self-correlated ambient noise, sampling rate drift between different devices, and choosing the best gesture.
- SG-LOC operates in the application layer and does not require any modification to the OS. We have implemented SG-LOC on commercial smartphones and performed an extensive evaluation.

1.3 Organization

The rest of this thesis is organized as follows. In Chapter 2 we describe a rogue AP detection technique for Wi-Fi networks. Then we present VR-Defender to defend against vehicular rogue APs for Drive-thru Internet in Chapter 3. In Chapter 4, we introduce DozyAP system to improve the power efficiency of Wi-Fi tethering. In Chapter 5, we present a localization algorithm to facilitate location-aware wireless communications among a group of devices. We conclude the dissertation in Chapter 6.

2 Client-side Rogue AP Detection

Security, energy management and location-awareness are three important aspects to enhance today's mobile technologies. In this opening chapter, we identify opportunities to improve mobile security through a particular problem: the detection of rogue APs. A rogue AP allows an adversary to launch a man-in-the-middle attack, thus posing a security threat to mobile clients. This chapter shows how to prevent clients from associating to malicious rogue APs without relying on any administrator or infrastructure support.

2.1 Introduction

The proliferation of IEEE 802.11 networks (WLAN) in public spaces such as airports and coffee houses has increased the interest of security and privacy when using such networks. A thread called *rogue access points*, or rogue APs, has emerged as an important security problem in WLANs [1--3, 57, 76, 86, 93]. A rogue AP is defined as an illegal access point that is not deployed by the WLAN administrator. Two types of rogue APs can be set with different equipments. The first type uses a typical wireless router connected directly into an Ethernet jack in a wall. The second type of rogue APs are set on a portable laptop with two wireless cards, one connected to a real AP and the other configured as an AP to provide Internet access to WLAN stations. We will further explore the differences between these two types of rogue APs in later sections. In this chapter we focus on the detection

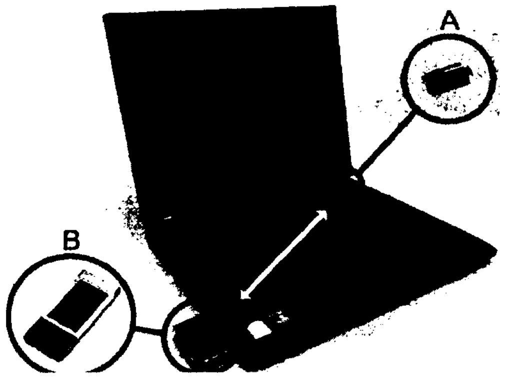


Figure 2.1: Demonstration of the hardware for setting up a rogue AP. The requirement is only a laptop with two wireless adaptors: A is an internal wireless adaptor which is connected to a legitimate AP, and B is an external wireless adaptor which behaves as a legitimate AP to induce users.

of the second type of rogue APs.

Fig. 2.1 illustrates that a laptop with two wireless adaptors can be easily set as the rogue AP considered in this chapter. For example, we let the internal wireless adaptor connect to a legitimate AP, and the external wireless adaptor pretend to be a real AP to induce users. In Linux, running command `iptables -t nat -A POSTROUTING -o interface -j MASQUERADE` can bridge packets from one adaptor to the other easily. According to 802.11 standard, when multiple APs exist nearby, a WLAN client will always choose the AP with the strongest signal to associate. To attract clients, therefore, a rogue AP needs to be close to clients so that its signal can be stronger than other legitimate APs. The rogue AP can then passively wait for users to connect to it, or actively send a fake de-associate frame to force users to change connection. Note that, the setting here only demonstrates the basic steps of setting up a rogue AP to launch attacks. In practice, a rogue AP needs further configuration to avoid easy detections, such as spoofing MAC address, SSID and vendor name, setting up a DHCP server to assign valid IP addresses to connected clients.

Once an innocent client is connected to a rogue AP, the adversary can manipulate and monitor the incoming and outgoing traffic of the client, and further launch different kinds of attacks. For instance, the adversary can easily launch phishing attacks by redirecting the user's web page request to a fake one to steal the user's sensitive information such as bank account and password.

The previous work has explored several approaches for rogue AP detection. One category of solutions is to measure some identities/fingerprints of an AP such as SSID, MAC address, RSSI, and clock skew. A rogue AP is detected when its identities are compared to those of legitimate APs. The other category of approaches is to analyze network traffic at the gateway to detect the presence of rogue APs (more details are described in Section 2.2). These existing approaches cannot effectively detect rogue APs from the client's side, especially in the strong adversary model considered in this chapter, where rogue APs are aware of the current detection schemes, and try to evade the detections. Here we list some challenges for designing a detection scheme:

- Clients may not have access to the information about legitimate APs, especially those APs deployed in hotspots. Therefore, it is not possible to compare the identity of an AP with that of authorized APs stored in the database.

- Clients have less privileges than administrators. They are limited by the settings of the network. For example, clients cannot set dedicated servers for detection. They cannot use the existing protocols that are not supported by the local network. Also, without the assistance from network administrators, it is not easy for clients to collect network traffic at the gateway. Therefore, some existing solutions cannot work.

- Malicious rogue APs know the detection schemes and manage to escape from detections. They can forge their identities, block or fake certain messages, and directly reply to clients without forwarding messages to legitimate APs. Therefore, simple defenses could be easily circumvented. In Section

2.3.2, we will look at the strategy for a sophisticated adversary in more detail.

In this chapter, we propose a timing-based rogue AP detection technique that allows end users to independently determine whether an AP is legitimate or not without the assistance from the WLAN administrators. To the best of our knowledge, we are the first to propose a rogue AP detection scheme that can be purely implemented by end users. Our main contributions are listed as follows:

1. We propose a timing-based rogue AP detection algorithm that is compatible with existing networking protocols and can be applied to 802.11 based network without further modifications.
2. Our solution can detect powerful rogue APs that manage to avoid detections as opposed to "accidental" rogue APs deployed, for example, by an innocent employee in an office [20].
3. We implement our scheme using commercial hardware and evaluate the performance in real-world networks. Extensive experiments show our algorithm achieves more than 90% accuracy in distinguishing rogue APs from legitimate APs in lightly-loaded traffic conditions and more than 60% accuracy in heavy traffic conditions.

The rest of the chapter is organized as follows. We describe the related work, and problem formulation in Sections 2.2 and 2.3 respectively. Our algorithm is detailed in Section 2.4, and our implementation is presented in Section 2.5. Finally, we discuss the evaluation results in Section 2.6 and conclude in Section 2.7.

2.2 Related Work

The threat of rogue APs has attracted significant attentions from both industrial and academic researchers. Existing rogue AP detections can be broadly classified into two categories.

The first category relies on special hardware such as wireless sniffers to monitor wireless network for rogue AP detection. These sniffers usually scan the spectrum at 2.4 and 5GHz for unauthorized traffic. The sniffers will alert the system administrators when such traffic is detected. Some commercial products [1--3] have been developed using this technique. In these products, a variety of identifying characteristics including MAC addresses, vendor name, and SSID are used to distinguish between a legitimate AP and a rogue AP. Other alternatives collect RSS values [75], radio frequency variations [26], and clock skews [46] as fingerprints to identify rogue APs. For example, work by [46] calculates every AP's clock skew by collecting their beacons and probe response messages. If any AP's clock skew is different from existing clock skews in the database, the AP is then identified as a rogue AP. Other work like [17, 20, 57, 93] proposes several hybrid detection schemes consolidating both wired and wireless-side efforts. For instance, in [93], special packets are sent to a specified wired station through wired network. If wireless sniffers capture such packets on air, the tested machine is identified as a rogue AP.

However, deploying wireless sniffers to adequately cover large scale networks such as public hotspots is very expensive. Our solution on the other hand has a much lower operating cost. In addition, our solution can be performed by the end users who have a natural interest in not connecting to a rogue AP, rather than relying on system administrators to disable the rogue APs.

In the second category, detecting the presence of rogue APs is achieved by analyzing network traffic at the gateway. In [24], the authors were among the earliest to suggest using temporal characteristics, such as inter-packet arrival time to detect rogue APs. Later work by [76] builds on this idea by creating an automated classifier. In [85] and [86], two similar detection schemes are proposed by examining the arrival time of consecutive ACK pairs in TCP traffics. Work in [84] and [58] utilizes the round trip time of TCP traffic to detect rogue APs, based on the CSMA/CA mechanism and physical properties of half duplex channel. Recent research [83] detects rogue APs by extracting characteristics unique to a wireless stream from network traffic.

The prior work focuses on detecting rogue APs that are directly connected to a wired network. Our detection scheme instead targets a different type of rogue AP attack, where the rogue AP is connected to legitimate APs. Furthermore, the prior work all considers detecting the rogue APs from network administrator's point of view, which is only feasible in corporate networks. In public networks like those found in coffee shops, users cannot assume that the network provider will implement any rogue AP detection scheme. Our proposed scheme can be executed by end users without any help from network administrators.

2.3 Problem Formulation

We consider a scenario when a wireless station tries to join a WLAN to access the Internet. After scanning the channels, the station will discover multiple APs within its communication range. Some of these APs are legitimate and some might be rogue APs. Our objective is to design an algorithm that helps the station to detect the rogue AP. The detection algorithm should function in all IEEE 802.11 based wireless networks without requiring additional modification. Our proposed scheme is a client-centric approach, where end users can perform the scheme on their own. It can be combined with administrator-centric approaches (described in Section 2.2) to reduce the risk of rogue APs.

We assume that the rogue AP will be launched using a mobile device with two wireless interfaces. The first interface connects the rogue AP to the legitimate AP. The second interface pretends to be a legitimate AP to induce users to connect to

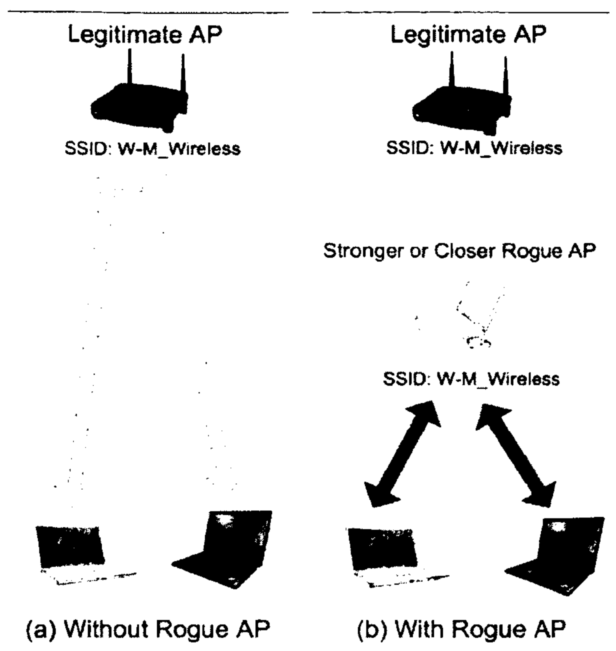


Figure 2.2: This figure shows the setup of a rogue AP. A rogue AP is connected to the wired network through a legitimate AP. Some stations inadvertently connect to the rogue AP because the rogue AP is closer and broadcasts stronger signals to them.

it. When a user associates to the rogue AP, the rogue AP will forward packets from the second interface to the first interface, and then towards the legitimate AP. This way, the user will still be able to access the Internet as if connected to a real AP. Fig. 2.2 illustrates the setup.

We do not consider a rogue AP setup where the adversary directly plugs the rogue AP into an Ethernet jack in the wall. There are three reasons for this.

First, there are a limited number of available Ethernet jacks in public places like airports. Since a rogue AP that needs an Ethernet cannot launch an attack without an available Ethernet jack, this makes this type of rogue AP attacks less likely in such places.

Second, rogue APs convince users to associate with them by offering a better connection as indicated by a stronger signal strength. Ethernet based rogue APs must remain connected to the Ethernet while launching the attack. As such, it is difficult for such rogue APs to physically move closer to users to increase their

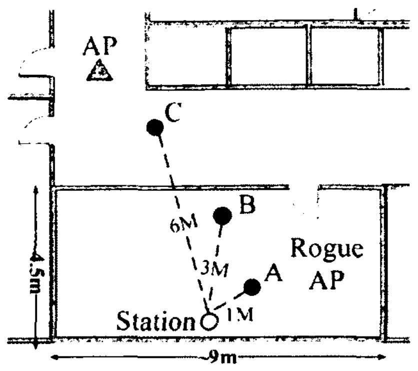


Figure 2.3: Illustration of rogue AP attacks. In our experiments, a rogue AP is deployed at location A, B and C.

signal strength to induce people to connect to them, thus limiting their impact.

Finally, network administrators can use other methods [17,20,83,86] to disallow devices from accessing the network via Ethernet jacks, if they are not registered or do not "behave" as wired stations. In this case, the rogue AP will be unable to provide Internet access to users, making them easy to be detected.

2.3.1 Rogue AP Effectiveness

To demonstrate the effectiveness of our rogue AP, we set up a testbed shown in Fig. 2.3. The station is placed in an office several meters away from a legitimate AP mounted on the ceiling. The SNR of legitimate AP measured by the station is 40dbm. We then setup the rogue AP and place it at three separate locations A, B, and C. Location A is one meter away from the station, location B is three meters away, and location C is 6 meters away behind a wall. The goal is to determine if we could induce the station to connect to the rogue AP instead of the legitimate AP. Table 2.1 shows the SNR values received by the station when the rogue is placed at different locations. By default, the station will select the AP with the highest SNR to connect. In our experiments, when the rogue AP's SNR is greater than 40 dbm, it is highly likely that the station will be lured into connecting with the rogue AP.

TX Power (dbm)	SNR (dbm)		
	A (1m)	B (3m)	C (6m through a wall)
18 (default)	71	55	40
14	67	51	36
10	61	47	33

Table 2.1: Average SNR under different distance and tx power

2.3.2 Adversary Model

Here we consider some defenses that can be circumvented by a sophisticated adversary.

Identity verification: Users can run programs like `traceroute` to determine whether the connected AP is a rogue AP, and `traceroute` will return the number of intermediate hops to a host site. From the output, the station will learn that a suspicious AP exists in the route.

However, the rogue AP can evade this detection by monitoring the wireless channel to learn the SSID and MAC address of a legitimate AP, and then setup the rogue AP to have the same parameters. The rogue can then avoid forwarding the real AP's reply to the user, thus giving the impression that it is connected to the same gateway as a legitimate AP.

Traffic monitoring: Traffic monitoring is a technique to distinguish between wireless and wired traffic. For instance, [86] monitors all the traffic at a gateway and computes the interval between two consecutive TCP ACK packets. A longer interval indicates that the TCP packets are traveling over a wireless connection.

However, since the user connecting to a legitimate or rogue AP must use a wireless link, the resulting interval between TCP ACKs will experience high variance due to fluctuating channel conditions. This makes the traffic monitoring technique unsuitable for rogue AP detection.

Simple timing: The station may use the timing information such as the round trip time (RTT) to detect a rogue AP. Since the rogue AP consists of an additional wireless link to the legitimate AP, this may lead to a delay when transmitting data. The station can determine the RTT by sending a message such as a `ping` request or TCP data packets [84] and wait for a reply.

However, the rogue AP can simply forge a response to the user, thus avoiding the time penalty of the additional wireless link. For instance, the rogue AP can generate a `ping` response to return to the user without forwarding the request to the real AP. Similarly when the user sends a TCP packet, the rogue AP can return the ACK to the user directly.

2.4 Our protocol

Our rogue AP detection protocol uses timing information based on the round trip time (RTT). The intuition is to let the user probe a server in the local network and then measure the RTT from the response. The user repeats this process for a number of times and records all the RTTs. If the mean value of RTTs is statistically larger than a certain threshold, we regard the associated AP as a rogue AP. We begin with examining the motivation and challenges of this approach, followed by some background discussion. We then propose our protocol and show how to determine the parameters.

2.4.1 Motivation and challenges

There are two reasons for using RTT-based method to detect rogue APs. First, when a user connects to the network via a rogue AP, all his packets traverse two wireless hops, one between the user and the rogue AP, and the other between the rogue AP and the real AP. When the user is communicating with a real AP, there is only one wireless hop. This additional hop will introduce an unavoidable time delay provided that the rogue AP is forced to communicate with the real AP. Second, it is easy for a user to measure RTTs. Unlike non-timing methods mentioned in the related work, measuring RTTs does not require any special equipment, such as sniffers [1, 2] or radio frequency analyzers [79]. It also does not require any modification to the AP.

However, using RTT to detect rogue APs requires addressing three issues:

(1) The first issue is which server to contact. A server in the local network is preferred over a remote server on the Internet because the RTT-based method is sensitive to the delay in the wired network. Probing a remote server may lead to significant variance of RTT due to the dynamic routing path and Internet traffic.

(2) The second issue is what type of probe message to use. We want a probe message that cannot be easily manipulated by the rogue AP, and can reach the server regardless of network configuration. As we mentioned earlier, a simple ping message can be easily returned by the rogue AP to evade detection and might be blocked by some network administrators. In addition, our probe message has to adhere to the existing networking protocols so as to avoid requiring assistance from the network provider.

(3) Finally, we have to consider the effect of network traffic conditions. A busy channel may adversely affect RTT timing and lead to incorrect rogue AP detection.

2.4.2 Background

Our solution lets the station contact a DNS server, and uses the DNS lookup as the probe message. In addition, we use two 802.11 management frames, probe request and probe response, to determine the effects of network traffic.

DNS server and lookup: The basic function of DNS is to provide a distributed database that maps human-readable host names (e.g., `www.cs.wm.edu`) to IP addresses (e.g., `128.239.26.64`). The servers managing this distributed database are known as DNS servers. Current networks typically cache the queried records to achieve high performance.

There are two typical types of DNS lookups: a *recursive* query and a *nonrecursive* query. In a recursive DNS lookup, a station queries a local server for a host name. If this server cannot answer the query, it will contact the root DNS server which will then recursively ask other servers to determine the IP address. In a nonrecursive query, the local DNS server will only search the cached records locally without contacting the root DNS server. If no matches are found, the local server will send a "host not found" message back to the station.

In our algorithm, we use nonrecursive query as the probe message to measure the RTT between the user and the DNS server. The user will send a DNS request for a host name with the nonrecursive option. The user then waits for the response from the local DNS server and measures the RTT. The user repeats this process using a different host name each time.

Our proposed scheme is efficient since most local networks may have a local DNS server or resolver for performance reasons [48]. Therefore, a station can always send a request to the local DNS server and the time spent on the wired network is small due to the local communication. Furthermore, since DNS lookup support is mandatory, all networks will have this function. Finally, since the DNS response varies for different queries, the adversary cannot predict in advance the user's query. The adversary also cannot determine whether a particular query can actually be satisfied by the real DNS server. Any rogue AP that returns an incorrect answer will be detected by the user. This forces a rogue AP to forward the request to the real DNS server to ensure that the reply is correct. Details of how to generate and verify user's queries are presented in Section 2.4.7.

Network traffic conditions: To determine the wireless traffic conditions, we measure another RTT using *probe request* and *probe response* messages. These messages are typically used when a station is scanning for APs.

There are two advantages of using probe request and response. First, by calculating the durations between these two packets, we can estimate the channel traffic and the AP's workload. The reason is that in a busy channel, both the probe request and response will take a long time to transmit due to channel contention and retransmission after signal collisions. Similarly, when the AP has a heavy workload, i.e., the AP is sending many packets for other associated stations, the probe response message has to wait in the AP's transmission queue for a long time before being sent out. Second, it is difficult for a rogue AP to replicate a busy channel by intentionally delaying the probe response because commercial wireless card drivers do not dispatch this kind of low level management frames to OS. Furthermore, it is difficult to delay a probe response since this function is not supported by regular wireless drivers.

However, a regular probe request has a drawback in that it is a broadcast message and every AP that overhears this request will respond. This leads to multiple responses, which will create unnecessary channel contention and lead to biased RTT measurements. Furthermore, a broadcast message will not be retransmitted if lost. The associated AP that does not receive the probe request correctly will never reply. This may affect the RTT values. Therefore, we modify the probe request packet to be a unicast message. This is done by putting the MAC address of the target AP into the destination field in the probe request. This will ensure that only the target AP will respond and other APs will not. Also, the station will automatically retransmit the probe request if needed.

2.4.3 Protocol overview

Here we present the overview of our rogue AP detection scheme. We use sta to indicate a station. For a given AP_x within sta's communication range, the station

Algorithm 1 Detecting Rogue AP (AP_x)

```
1: Connect and associate with $AP_x$
2: for $i = 1$ to $n$ do
3:   Send unicast probe request to $AP_x$, record round trip time $RTT_{probe} = RTT_{probe} - T_{data}(probe)$.
4:   Send DNS lookup to local DNS server, record round trip time $RTT_{dns} = RTT_{dns} - T_{data}(dns)$.
5: end for
6: Filter out outliers
7: $\overline{RTT_{probe}}$ = Mean of remaining $RTT_{probe}$
8: $\overline{RTT_{dns}}$ = Mean of remaining $RTT_{dns}$
9: $\sigma_{probe}$ = Standard deviation of remaining $RTT_{probe}$
10: $\sigma_{dns}$ = Standard deviation of remaining $RTT_{dns}$
11: $\Delta t = \overline{RTT_{dns}} - \overline{RTT_{probe}}$
12: $\theta = f(\sigma_{probe}, \sigma_{dns})$
13: if $\Delta t > \theta$ then
14:   $AP_x$ is a rogue AP
15: end if
```

runs the Algorithm 1 to determine whether AP_x is a rogue AP.

Our algorithm consists of two phases. The first phase (lines 2-5) measures the RTTs, and the second phase (lines 6-15) analyzes the collected RTTs and decides if the tested AP is rogue.

In the first phase, the station repeatedly sends a probe request (line 3) and a DNS lookup (line 4) for n rounds. RTT_{probe} (RTT_{dns}) records the round trip time between the probe (DNS) request and response. Note that we subtract the data transmission time T_{data} from both RTT_{probe} and RTT_{dns}, because probe packets and DNS packets have different packet sizes and transmission rates which may vary in each round. After eliminating the effects caused by data transmission time, we can compare RTT_{probe} and RTT_{dns} fairly. The detail of how to calculate T_{data} is discussed in next subsection. The choice of parameter n captures the tradeoff between the overhead and accuracy. Larger n incurs a larger overhead, but increases the detection accuracy. We describe this issue in Section 2.4.6.

In the second phase, we first filter out some outlier RTTs (line 6). After that, we calculate the mean value (lines 7-8) and standard deviation (lines 9-10) of both RTT_{probe} and RTT_{dns}. Finally, in lines 11-15, we check the difference between these two RTTs Δt (line 11) against a threshold parameter θ (line 12) to determine whether this is a rogue AP. The threshold θ reflects the delay induced by the extra wireless transmissions in rogue AP case, and is calculated by σ_{probe} and σ_{dns} according to experimental measurements. We present the detail of function f in Section 2.4.5.

2.4.4 Outlier filter

As mentioned earlier, after measuring n sample values of the RTTs, our algorithm runs a filtering process to eliminate some abnormally large values of RTTs that may exist in the n samples due to dynamic network conditions. We call these abnormal values *outliers*. These outlier values may affect the final outcome. For example, assuming we have 50 RTT samples, and 49 samples are 1ms and a single sample is 100ms. Without filtering out 100ms sample, we arrive at a mean value of 2.98ms, which is not representative of the majority of the samples.

There are many ways to define an outlier. In this chapter, we consider a conventional definition based on the value distance. Consider n sample values are illustrated in a one-dimension space and each value is represented by a data point. The distance between any two data points is defined as the absolute difference between their values. Let $D^k(p)$ represent the distance between data point p and its k-th nearest neighbor point, an outlier is defined as follows: *Given k and m, we first sort all n samples according to the value of $D^k(p)$. The data points with the top m largest values of $D^k(p)$ are called outliers.*

Algorithm 2 illustrates the filtering process. We set $k = m = 0.2 \cdot n$, where n is the number of samples, i.e., we will filter out 20% abnormal values.

Algorithm 2 k-nearest neighbors outlier filter

```
1: $k = m = 0.2 \cdot n$
2: for $i = 1$ to $n$ do
3:     $d_i = D^k(RTT_i)$
4: end for
5: Sort all $d_i$ in increasing order
6: Remove the top $m$ largest values
```

Parameters	Values		
	802.11b	802.11g	802.11a
t_{slot}	20 μs	9 μs	9 μs
t_{DIFS}	50 μs	34 μs	28 μs
t_{PCLP}	192/96 μs	192/96 or 20 μs	20 μs
CW_{min}	31	15	15

Table 2.2: IEEE 802.11 characteristics

2.4.5 Parameter values

Here we explain how to derive T_{data} and θ used in Algorithm 1. We begin with a quick review of the 802.11 protocol.

IEEE 802.11 medium access control adopts CSMA/CA model and the distributed coordination function (DCF). Before transmitting a frame, a station first senses whether the channel is idle. If the channel is idle, the station will transmit immediately. Otherwise the frame transmission will be deferred until the channel becomes available. After the channel is free for certain period of time, which is defined as DIFS, the station starts a back-off operation with a slot counter whose value is randomly selected between 0 and the size of a contention window (CW_{min}). This back-off counter decreases by one for each idle slot time. When the back-off counter becomes zero, the station transmits the frame. When the destination receives the frame successfully, it sends a MAC layer ACK back to notify the sender. If the sender does not receive an ACK, it will retransmit the packet. Table 2.2 lists some timing parameters in 802.11 standard that we will use later.

Based on 802.11 mechanism, we can express the delay for transmitting a packet as

$$T_{delay} = t_{defer} + t_{DIFS} + t_{bf} + T_{data} + t_{retransmit},$$

where t_{defer} is the time deferred due to a busy channel medium, $t_{retransmit}$ is the time for retransmission if no ACK is received, and t_{bf} is the random back-off time. The expected value of t_{bf} is given by

$$\overline{t_{bf}} = \frac{CW_{min}}{2} \cdot t_{slot}.$$

Data transmission time T_{data} depends on the data size (L-byte payload) and the transmission rate (r Mbps),

$$T_{data}(L, r) = t_{PCLP} + \frac{(28 + L) \cdot 8\text{bits}}{r},$$

where t_{PCLP} is the physical layer packet overhead of any IEEE 802.11 packet including two parts: the Physical Layer Convergence Protocol (PCLP) preamble used for synchronization and the PCLP header. According to the standard, T_{PCLP} is $192\mu s$ ($96\mu s$) for long (short) preamble, using ERP-DSSS modulation scheme (supporting 1-11 Mbps). The T_{PCLP} is $20\mu s$, using ERP-OFDM modulation scheme (supporting 6-54 Mbps) [81]. The value 28 is the length of MAC header plus CRC checksum. For every measured RTT at each round, the station is aware of the data size (L) and transmitting rate (r) for every incoming and outgoing packet. The station can thus compute the exact values of $T_{data}(probe)$ and $T_{data}(dns)$, and subtract them from RTTs to eliminate the effect of different transmission time.

In order to derive θ, we need to analyze the RTTs. For a legitimate AP, the path taken for an entire probe request and response is

$$STA \rightarrow AP \rightarrow STA,$$

and the path taken for an entire DNS lookup and answer is

$$STA \rightarrow AP \rightarrow SERV \rightarrow AP \rightarrow STA.$$

For simplicity, we only consider the network overhead, but ignore the time for AP and DNS server to process the packets. There, after subtracting T_{data}, these two RTTs for probe and DNS can be expressed as

$$RTT_{probe} \approx T_{overhead}^{sta \rightarrow ap} + T_{overhead}^{ap \rightarrow sta}$$

and

$$RTT_{dns} \approx T_{overhead}^{sta \rightarrow ap} + T_{wired} + T_{overhead}^{ap \rightarrow sta},$$

where $T_{overhead}$ is used to indicate the remaining part of T_{delay} after deducting T_{data}. Since the RTTs of DNS and probe are measured at approximately the same time, we assume the network conditions are stable during that time period[1], so we can regard $T_{overhead}$ of probe and DNS as the same. Thus, the difference between two RTTs is

$$\Delta t = \overline{RTT_{dns}} - \overline{RTT_{probe}} = T_{wired}.$$

Based on our extensive experimental measurements (which will be shown later in Section 2.6), Δt is no larger than 1.3 ms, even when we consider several hops between sending a DNS lookup and receiving the answer back in the idle network traffic condition.

On the other hand, if the tested AP is a rogue AP, the path taken for probe

[1] [88] mentions that wireless network traffic remains stable within approximately $150 - 250$ ms in practice.

messages is still

$$STA \rightarrow RAP \rightarrow STA,$$

but the path for DNS messages is

$$STA \rightarrow RAP \rightarrow AP \rightarrow SERV \rightarrow AP \rightarrow RAP \rightarrow STA.$$

Similarly, we get

$$\begin{aligned}\Delta t' &= \overline{RTT_{dns}} - \overline{RTT_{probe}} \\ &= T_{delay}^{rap \rightarrow ap} + T_{wired} + T_{delay}^{ap \rightarrow rap}.\end{aligned}$$

For a station, it is difficult to estimate T_{delay} between the rogue AP and its associated legitimate AP, since the station does not know the transmission rate and network condition at the AP side. However, based on our experiments, we observe that $\Delta t'$ is larger than 1.3 ms. In order to effectively detect a rogue AP, θ needs to be set between Δt and $\Delta t'$ as $\Delta t < \theta < \Delta t'$. Therefore, we set the threshold θ to be 1.3 ms.

However, the threshold θ may not perform well under heavy traffic condition. This is because that the heavy traffic congests the AP causing long tx-queue, packet loss, and retransmission. The mean value and variance of RTTs for both probe and DNS will become larger. Additionally, heavy workload may delay each packet waiting in the AP's queue until packets buffered ahead are transmitted.

To overcome this problem, we dynamically adjust the threshold θ according to the standard deviation of RTT_{probe} and RTT_{dns}. Based on our extensive experiments, we find that the standard deviation is a good indicator of traffic load. Heavy traffic conditions usually result in larger deviation than light traffic conditions. Fig. 2.4 illustrates the mean value of Δt against the average value of standard devi-

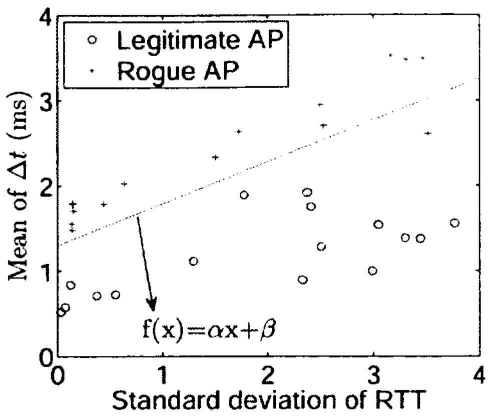


Figure 2.4: Mean value of Δt against the average value of standard deviation of RTT_{probe} and RTT_{dns} under different traffic load condition

ation of RTT_{probe} and RTT_{dns}. In the figure, each data point indicates a test case for a legitimate AP or a rogue AP. They are plotted with respect to their mean values of Δt and their standard deviations of RTT. As shown in the figure, we can divide the data points into two groups, one for legitimate APs and the other for rogue APs, by a single line. From the data, we empirically set

$$\theta = f(\sigma_{probe}, \sigma_{dns}) = \alpha \cdot \frac{\sigma_{probe} + \sigma_{dns}}{2} + \beta, \tag{2.1}$$

where the α and β values are 0.49 and 1.3 respectively. We derive these parameters by adjusting the line to let the most number of data points of legitimate APs above the line, and the most number of data points of rogue APs below the line. The particular line with the furthest mean distance to each data point is selected.

Finally, recall in Algorithm 1, after measuring enough samples of RTTs, a station computes the Δt and θ. If $\Delta t > \theta$, the station will mark the AP it connects to as a rogue AP. Then the station will choose another AP for test.

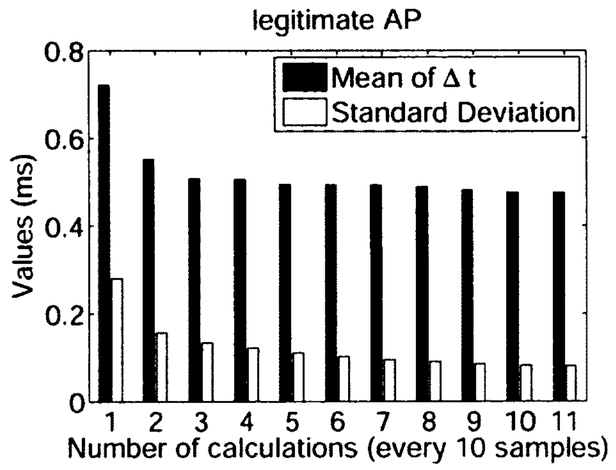


Figure 2.5: Illustration of the trend of mean value and standard deviation of Δt against the number of samples.

2.4.6 Number of samples

In our experiments, we find that when the wireless traffic is lightly-loaded, our detection algorithm can detect rogue AP with high probability even if we use a small number of samples. However, when the traffic is heavy, the algorithm needs more samples to achieve desired accuracy. In order to reduce the detection time without sacrificing accuracy, we present a heuristic algorithm to adjust the number of samples dynamically, rather than using a fixed number.

The intuition is based on the experimental observations. In Fig. 2.5, we illustrates the trend of the mean and standard deviation of Δt against the sample size. In the horizontal axis, each pair of two bars present the mean value of Δt and the standard deviation of RTT every 10 samples. We observe the values may vary a lot if the sample size is not large enough. But once collecting enough number of samples, the variance will be stable. Therefore, to check whether the number of samples is large enough, we re-calculate the variance of the whole samples every 10 samples, and compare it with previous values. When the difference of the variances is smaller than a predefined threshold, we stop sampling. This is because additional samples will not help detection.

2.4.7 DNS operations

Our scheme has two DNS operations. The first is to determine a set of n different host names for measuring n samples of RTTs. The second operation is verifying DNS answers.

Determining DNS queries. We generate DNS queries as follows. In a station, two pools are constructed. The first pool contains valid host names that can be extracted from local caches of web browsers (like firefox and IE). The other pool contains some randomly generated host names. We do not know whether they are valid or not. Once the two pools have been constructed, we will randomly select a pool to pick a host name to test. Then, we delete that host name from the pool to avoid using it again. This prevents a rogue AP from remembering the corresponding answers. Note that if we need a lot of samples, we assign a smaller weight to the first pool to prevent it from exhausting too fast.

Verifying DNS answers. Suppose that a station hears $m + 1$ $(m \geq 1)$ APs composed of m legitimate APs and one rogue AP. The station will first randomly select one AP and send a recursive DNS query to it.

Assuming this selected AP is *not* a rogue AP, it will execute this recursive query. This forces the local DNS server to provide an answer to the query by querying other name servers on the Internet and cache the response. The station then uses the same host name and queries all other APs in non-recursive queries. Now the answer to the query should be cached on the DNS server. We then execute Algorithm 1 on the remaining APs. The legitimate APs will respond accordingly with a reasonably short RTT. For the rogue AP, if it chooses to forward the query, the rogue AP will be detected by our algorithm. If the rogue AP does not forward the query, the rogue AP does not know the correct answer and can only return a "host not found" message. The station can thus determine that AP is a rogue AP.

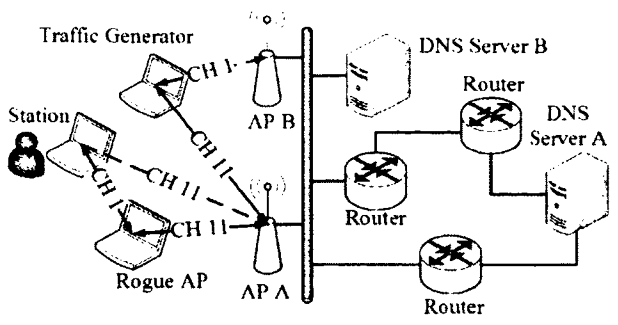


Figure 2.6: Illustration of the architecture of the testbed

If the selected AP *is* a rogue AP, it must forward the recursive DNS query to the real AP. This is because if the rogue AP does not forward the query, the DNS server would not contain the correct answer. When the station runs our algorithm, all the other APs will reply with a "host not found" message. When this happens. the rogue AP will be detected, since a legitimate AP will always execute the recursive DNS query.

We then repeat the process for the remaining APs to detect the rogue AP. In this chapter, we do not consider the case of multiple rogue APs colluding with each other. This will be examined in our future work.

2.5 Implementation and Setup

This section describes the experiment setup and system implementation in detail.

2.5.1 Hardware description

Fig. 2.6 illustrates the infrastructure of our testbed which consists of two APs and three laptops: one laptop is used as a traffic generator, and the remaining laptops serve as a station and a rogue AP. Server A is the DNS server in the campus network. To investigate the effect of wired link on our algorithm, another DNS server B in the same subnet of APs is utilized.

Commercial Wireless Cards	Chipset	Driver
Intel PRO/Wireless 3945ABG	Intel	ipw3945 (v1.2.2)
TP-Link TL-WN610G	Atheros	madwifi (v0.9.4)
D-Link WNA-2330	Atheros	madwifi (v0.9.4)
Dell wireless 1390 WLAN	Broadcom	BCM4311

Table 2.3: Description of commercial wireless cards, chipsets and corresponding drivers used in our experiments

The hardware specification of each component is described as follows.

Access Points: We use a Linksys WPA54G and D-Link DI-624+A for our APs. Both APs operate in the 802.11g mode.

Wireless Stations: All laptops in Fig. 2.6 are 2GHz x86 machines running Linux 2.6x kernel. The traffic generator and the station are equipped with a TP-Link TL-WN610G wireless card while the rogue AP possesses 2 wireless cards, one TP-Link TL-WN610G, and the other Intel 3495ABG.

DNS Server: Both DNS servers are campus servers connected in local wired networks at different locations.

2.5.2 Software description

Drivers: We use Madwifi (v0.9.4) driver [11] for the wireless cards with Atheros chipset, ipw3945 (v1.2.2) driver [9] for those with Intel chipset, and BCM4311 linux driver for those with Broadcom chipset. Table 2.3 lists all wireless cards, chipsets and corresponding drivers used in our experiments.

Click toolkit on station: On the station side, we implement the proposed algorithm using Click [49] toolkit with the wireless card turned into monitor mode. Click toolkit is a powerful tool over the driver layer. It is well connected with the wireless card's monitor mode and provides a flexible programming environment to implement our protocol. The most important feature is that we can inject raw data by using Click, such that our unicast probe request can be sent easily. The

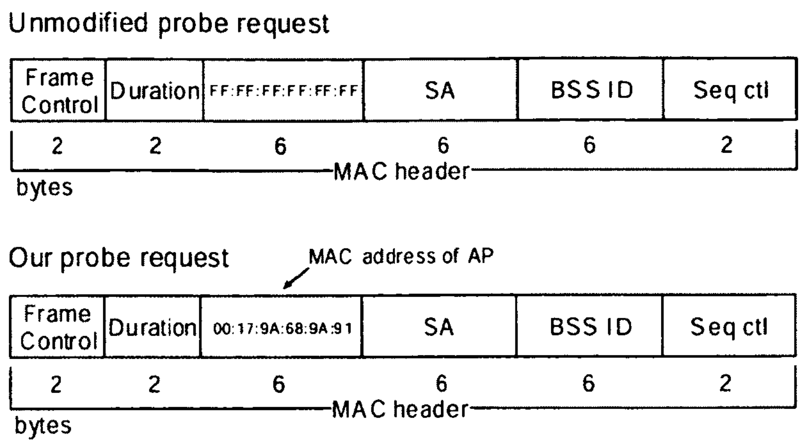


Figure 2.7: MAC header comparison between unmodified probe request and our probe request

comparison between two probe requests are shown in Fig 2.7.

Configuration on the rogue AP: For the rogue AP, one of its wireless cards is configured to work in the AP mode, and the other wireless card is configured to the station mode and connects to a legitimate AP. Tunneling these two interfaces is achieved by adding rules in *iptables*.

We use a tool called *macchanger* to spoof the MAC address of a legitimate AP. A station connected to the rogue AP will be assigned a valid IP address by *dnsmasq*, as if it obtains it from a legitimate AP. The adversary's strategies mentioned before are implemented by *netfilter/iptables*.

Traffic load: We use channel utilization as in [47] to quantify the traffic load. The channel utilization per second is computed by adding (1) the time spent by the on-air transmission of all data (including retransmitting), management, and control frames transmitted during a second, and (2) the overhead for each frame, such as DIFS and SIFS. This overhead is a part of channel utilization, since the channel remains unavailable at that time. In our experiments, the traffic generator will send packets with constant bit rate (CBR) to generate a required channel utilization.

Recording RTTs: Click toolkit leverages libpcap [10] to push/pull packets to/from the WLAN driver (shown in Fig. 2.8). Once a probe request (or DNS lookup)

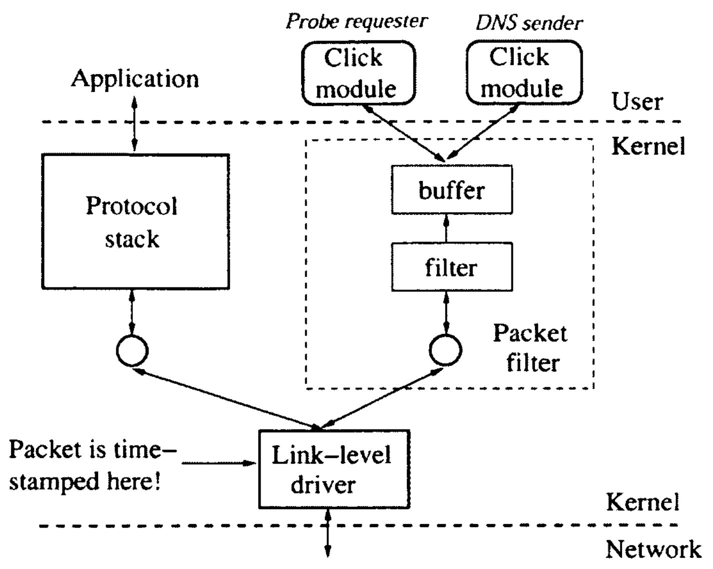


Figure 2.8: Basic journey of a packet and the place where we record RTTs.

is created in a click module, the packet will be sent to the WLAN driver for link-level processing before being pushed to the tx-queue of the wireless card. We record the transmission time just before the packet is pushed to the tx-queue. It is known this time is not the instance when the packet is actually transmitted, since the packet must wait until previous packets in the queue have been transmitted. To account for this delay, we regulate the rate in which packets are added to the tx-queue such that there is at most one packet in the queue at all times.

When the wireless card receives a probe or DNS response, we record the arrival time when an interrupt is delivered to the driver. This may incur a slight delay since the kernel has to process the interrupt. Since we determine Δt by subtracting RTT_{probe} from RTT_{dns}, this delay is eliminated.

Unlike previous method mentioned in [40] where RTTs are recorded in user space, our method can record more accurate RTTs. That is because each packet is timestamped close to the time when the packet is actually being sent or received. This may prevent including the delay for packets walking through the kernel. Thus, our measured RTTs are not affected by the workload of the station.

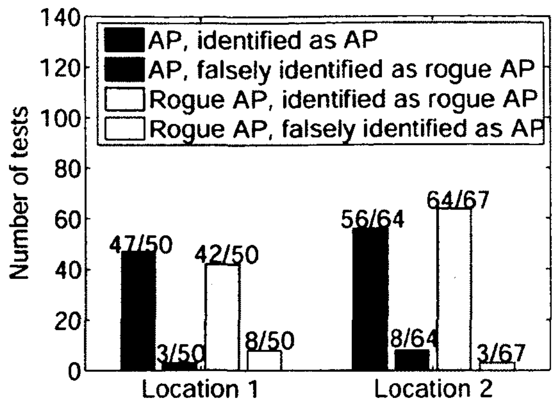


Figure 2.9: Testing the accuracy of our detection algorithm at different locations in real world. Location 1 is in the China State Key laboratory of Novel Software Technology at Nanjing University, and location 2 is in the campus at College of William and Mary.

2.6 Evaluation

Here we present the experimental results of our rogue AP detection algorithm. We use real settings to evaluate the robustness of our algorithm in practice. Our experiments are performed in two campuses referred to as location 1 and location 2. The configurations in both places follow similar architecture shown in Fig 2.6. However, the network environment including AP's capacity, the workload of local DNS server, the number of users, and interference may be different.

Fig. 2.9 shows the observable experience of our detection algorithm in those two places. In the location 1, we first use the algorithm to test a real AP 50 times. Our approach only fails 3 times. The false positive rate is only 6%. Then, We repeat tests for determining a rogue AP. At this time, detection fails 8 times. The false negative rate is nearly 16%. Similar experiments are conducted in the location 2. The corresponding false positive rate and false negative rate are 12.5% and 5% respectively. As we see, our detection accuracy is about 90% in total. That is really robust in practice.

In the following, we investigate the performance of our detection scheme, while considering some factors that may affect our algorithm. Recall that, the key of our approach is using the round trip time to detect a rogue AP. We consider the following factors that have influence on timing RTT.

Data transmission rate: RTT is inversely proportional to data transmission rate. High transmission rate usually leads to small RTT. In Section 2.6.1, we investigate whether a rogue AP can manipulate its transmission rate to avoid detection.

Location of DNS server: In some small hotspots (eg. coffee shops, restaurants), APs are usually connected to a close DNS server or resolver provided by ISP. This server may be located some hops away from APs. In this case, we have possibility to falsely identify a legitimate AP as a rogue AP due to large RTT. Section 2.6.2 describes the impact of this factor. We show that our scheme can tolerate a DNS server with several hops away.

Wireless traffic: As mentioned early, wireless traffic may incur large variance of RTT. That is because some packets may be sent immediately with no contention, but some packets may be deferred for a long time due to collision or interference with others. The variance may hide rogue AP's additional wireless link, and make the detection hard. In Section 2.6.3, we evaluate our algorithm under different wireless traffic conditions.

AP's workload: AP's workload is related to the utilization of AP's queue. It is caused by network traffic, but not equivalent to the traffic. We examine this factor in Section 2.6.4.

Finally, Section 2.6.5 discusses the accuracy of our algorithm by using different number of samples, and Section 2.6.6 shows how much time we will spend to test an AP.

RAP	Average RTT after removing outliers (ms)					
Rate adaptation	$\overline{RTT_{probe}}$	0.72	0.73	0.73	0.73	0.72
	$\overline{RTT_{dns}}$	2.66	2.71	2.65	2.67	2.66
	Δt	1.94	1.98	1.92	1.94	1.94
	θ	1.33	1.32	1.32	1.33	1.33
Fixed 54Mbps	$\overline{RTT_{probe}}$	0.73	0.72	0.73	0.74	0.73
	$\overline{RTT_{dns}}$	2.92	3.13	2.98	3.92	2.84
	Δt	2.19	2.41	2.25	3.18	2.11
	θ	1.32	1.43	1.33	1.44	1.32

Table 2.4: Comparison between rate adaptation and fixed 54Mbps under idle traffic condition

2.6.1 Data transmission rate

Most wireless devices adopt rate adaptation algorithms to adjust their transmission rate with respect to varied wireless conditions. However, since there are no specifications with regards to rate adaptation in 802.11 networks, the rogue AP is free to use any 802.11 transmission rate to try to avoid detection. The idea is that a rogue may attempt to always use the highest rate when connected to a legitimate AP so as to reduce the RTT.

To test, we first set up a rogue AP to use the default rate adaptation in idle traffic condition, and run our detection algorithm. We then repeat the experiment using the same traffic condition, except we set the rogue AP to always use the highest possible transmission rate of 54Mbps. In both tests, we use the same settings, where sample size $n = 100$ and contacting the DNS server B. Table 2.4 shows the results for the two tests, where $\overline{RTT_{probe}}$ (or $\overline{RTT_{dns}}$) is average RTT between probe (or DNS) request and the response minus the data transmission time (see lines 7-8 in Algorithm 1), $\Delta t = \overline{RTT_{dns}} - \overline{RTT_{probe}}$, and θ is computed according to Eq.(1). In our algorithm, if $\Delta t > \theta$, the tested AP is identified as rogue; otherwise as legitimate. We observe that (1) even if the rogue AP were set to always send at the highest possible rate, we can still detect the rogue AP, and (2) the performance

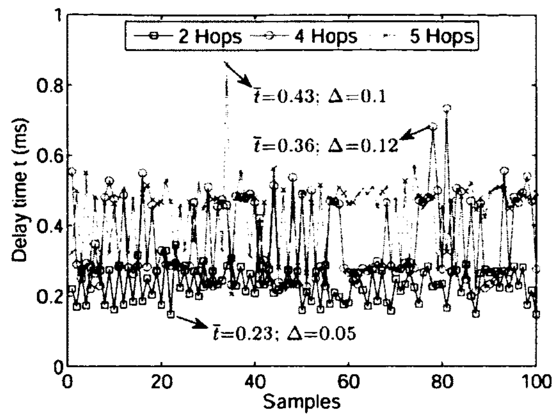


Figure 2.10: Delay for transmitting 64 byte packets via different number of hops on wired link. $\bar{t}$ is the mean of delay, and Δ is the variance.

gain by the rogue AP in using a fixed rate appears to be minor, since rate adaptation can quickly converge to use the best possible rate even if the initial rate is much lower. In fact, in a practical environment, using a fixed rate may result in a *worse* performance since more packets will be dropped when traffic conditions fluctuate. This is shown in the larger Δt values in fixed rate experiments. These results are omitted due to page limit. Lastly, since utilizing a fix rate yields no benefits, we let the rogue AP use rate adaptation for the rest of our experiments.

2.6.2 Location of DNS server

To illustrate the delay introduced by multiple wired hops, we send 64 byte packets to a local host located at two, four, and five hops away from the station, and measure the time taken for the host to respond. Fig. 2.10 shows the results. We find that the increased time resulting from additional hops is every small. One additional hop only incurs less than 0.1 millisecond time.

Next, we examine our detection algorithm when tested APs are connected to different DNS servers under idle traffic condition. In the first test, we let the legitimate AP and the rogue AP both use a far DNS server A (see Fig. 2.6). Packets sent by the station need 3 wired hops to reach the DNS server, and 2 wired hops

	Average RTT after removing outliers (ms)					
Legitimate AP (Server B)	$\overline{RTT_{probe}}$	0.633	0.633	0.632	0.634	0.634
	$\overline{RTT_{dns}}$	1.152	1.151	1.148	1.156	1.130
	Δt	0.519	0.518	0.516	0.522	0.025
	θ	1.314	1.315	1.316	1.316	1.503
Rogue AP (Server B)	$\overline{RTT_{probe}}$	0.671	0.663	0.664	0.664	0.661
	$\overline{RTT_{dns}}$	2.151	2.219	2.452	2.371	2.455
	Δt	1.48	1.556	1.788	1.707	1.794
	θ	1.357	1.345	1.357	1.362	1.362
Legitimate AP (Server A)	$\overline{RTT_{probe}}$	0.73	0.0.72	0.73	0.73	0.73
	$\overline{RTT_{dns}}$	1.59	1.60	1.58	1.62	1.67
	Δt	0.86	0.89	0.85	0.89	0.94
	θ	1.33	1.36	1.33	1.33	1.35
Rogue AP (Server A)	$\overline{RTT_{probe}}$	0.74	0.74	0.74	0.74	0.75
	$\overline{RTT_{dns}}$	2.76	2.92	2.71	2.67	2.84
	Δt	2.02	2.18	1.97	1.93	2.09
	θ	1.41	1.43	1.39	1.41	1.41

Table 2.5: Average RTT for DNS server under idle traffic situation

for the response coming back. In the second test, we have both legitimate and rogue APs connect to a close DNS server B which is located in the same subnet with APs. Table 2.5 shows the results. We see that our algorithm is able to detect the rogue AP even when the DNS server is located at the place several hops away.

2.6.3 Wireless Traffic

Here, we examine the effects of wireless traffic on our detection algorithm. Since we adopt a timing-based approach, variations in network traffic may adversely affect our results. To only evaluate the negative impact, we ignore the traffic occurs on the channel used between rogue AP and real AP, since this will help us to detect the rogue AP. We only consider the wireless traffic between the station and the tested AP, and set the rogue AP to use the most favorable conditions to avoid detection. Because the rogue AP can best avoid detection when it can forward packets from the station to the real AP as fast as possible, we let the connection between the rogue AP and the AP A be free of any traffic, thus ensuring the fastest

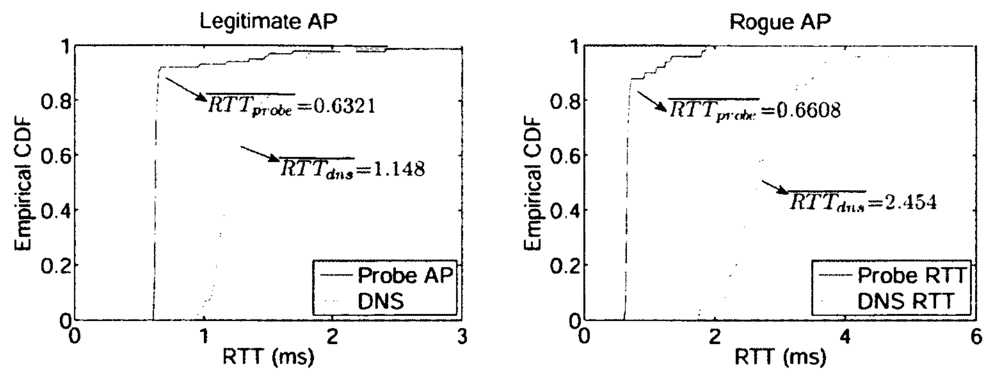


Figure 2.11: Empirical CDF of RTT for legitimate AP A and rogue AP in idle traffic situation, while the DNS server is B, transmission rate is automatic, and $n = 100$.

possible transmission between the rogue AP and real AP. In our experiments, this connection is set to the channel 11 with idle traffic and good quality of signals. We then test the rogue AP against AP B, both of which are set to the channel 1. We use separate laptops as a traffic generators to control the amount of traffic on that channel. We experiment over three traffic conditions, idle traffic, half-saturated traffic, and saturated traffic. In all experiments, we set $n = 100$ and use DNS server B.

Idle traffic: We create idle traffic condition by restricting data packets on the channel. But note that, idle traffic does not mean the channel utilization (as mentioned in Section 2.5.2 (d)) is zero (nearly 10% in our testbeds), since there are management frames including beacons and so on. Fig. 2.11 describes the empirical CDF of RTT for a legitimate AP and a rogue AP measured in one experiment. The complete results are listed in Table 2.5. As we can see, the value of Δt is small, and the θ varies a little in idle traffic situation. For the legitimate AP, all Δt are smaller than corresponding θ, whereas all Δt for the rogue AP are larger than θ. Our scheme achieves nearly 100% accuracy, and the total testing time is no longer than 1s.

Half-saturated traffic: We define a half-saturated traffic condition when the ratio of on-air time of all transmitted packets to the total time is nearly 45%. The

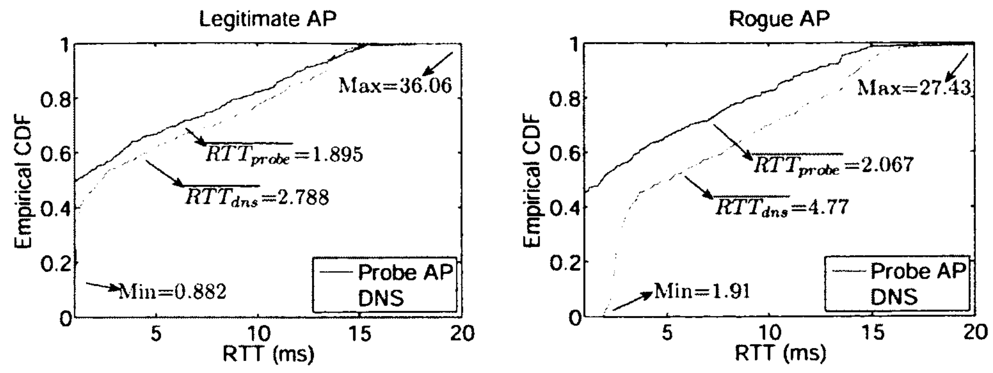


Figure 2.12: Empirical CDF of RTT for legitimate AP B and rogue AP in half saturated traffic situation, while DNS server is B, transmission rate is automatic, and $n = 100$.

	Average RTT after removing outliers (ms)					
Legitimate AP (Server B)	$\overline{RTT_{probe}}$	1.895	1.67	2.009	1.664	1.787
	$\overline{RTT_{dns}}$	2.788	3.486	3.291	3.415	3.133
	Δt	0.893	1.816	1.282	1.751	1.346
	θ	2.556	2.572	2.825	2.586	2.585
Rogue AP (Server B)	$\overline{RTT_{probe}}$	1.744	2.067	2.69	2.749	1.982
	$\overline{RTT_{dns}}$	4.692	4.77	6.215	6.222	4.838
	Δt	2.948	2.703	3.525	3.473	2.856
	θ	2.821	2.634	2.896	3.156	2.672

Table 2.6: Average RTT in half saturated traffic situation

traffic generators periodically send packets to create this condition. The experiment is then repeated to test our algorithm. We find that as the traffic load increases, the average RTT for both probe and DNS messages also increase. At the same time, our algorithm is still able to identify the rogue AP with high probability. Fig. 2.12 illustrates CDF for one experiment. The details are shown in Table 2.6.

Saturated traffic: Here, we let the traffic generators send enough packets to create a 90% channel utilization before starting the experiments. Fig. 2.13 and Table 2.7 describe the results. We find that under heavy traffic condition, the variance of RTT for a probe request (DNS lookup) becomes large. The values range from several milliseconds to hundreds of milliseconds. As a result, some of the legitimate APs may be incorrectly classified as a rogue AP.

In summary, Fig. 2.14 shows the trends of the mean value of Δt and the thresh-

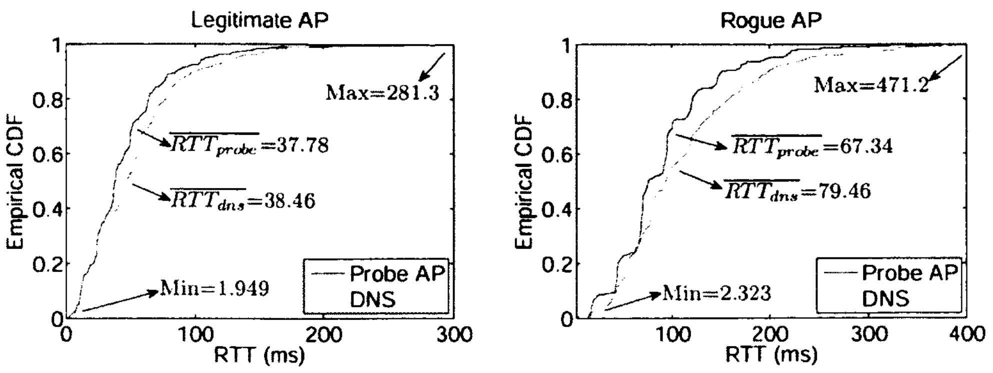


Figure 2.13: Empirical CDF of RTT for legitimate AP B and rogue AP in saturated traffic situation, while DNS server is B, transmission rate is automatic, and $n = 100$.

	Average RTT after removing outliers (ms)					
Legitimate AP	$\overline{RTT_{probe}}$	37.78	41.07	52.55	54.63	**29.36**
	$\overline{RTT_{dns}}$	38.46	45.22	61.28	64.43	**41.72**
	Δt	0.68	4.15	8.73	9.80	**12.36**
(Serv B)	θ	7.70	11.57	14.12	13.49	**9.56**
Rogue AP	$\overline{RTT_{probe}}$	53.04	63.18	67.34	59.18	**54.95**
	$\overline{RTT_{dns}}$	69.13	82.34	79.46	77.59	**69.36**
	Δt	16.09	19.16	12.12	18.41	**14.41**
(Serv B)	θ	12.42	13.29	11.73	12.86	**15.33**

Table 2.7: Average RTT in saturated traffic situation

old θ for both legitimate AP (a) and rogue AP (b) varying against the channel utilization. In the experiments, we set $n = 100$ and contact the DNS server B. In the figure, the circles represent the false detections. As we see, the threshold θ according to the standard deviation of RTT can reflect the traffic condition. Large channel utilization incurs a large value of θ. By dynamically adjusting the threshold, our detection algorithm can achieve relatively high accuracy under different channel utilization.

2.6.4 AP's workload

Besides the channel contention of wireless traffic, AP's workload also affects packet's RTT. When an AP suffers heavy workload, each packet needs more time to process, and has to wait in the queue of the AP until packets ahead are sent out. It will adversely affect the detection. Hence, we focus on AP's workload to evaluate

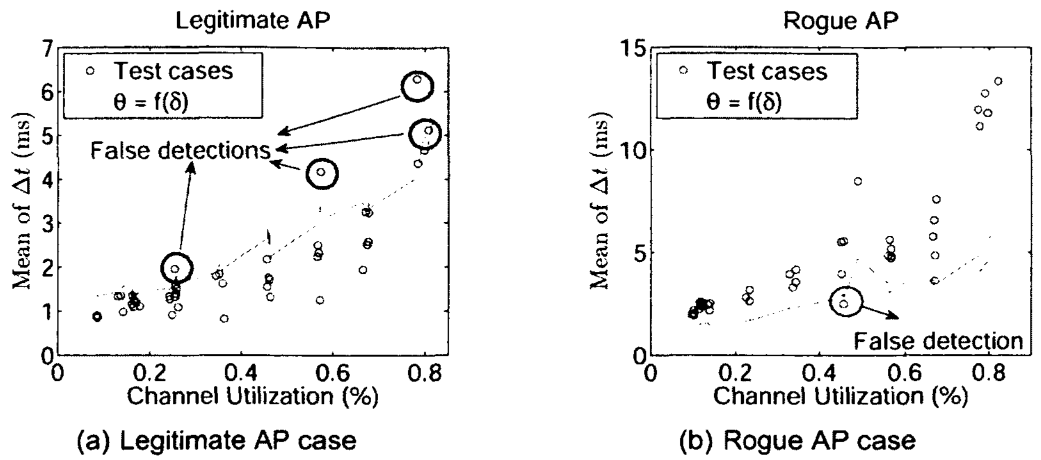


Figure 2.14: Illustration of the trend of the mean value of Δt and the threshold θ varying against the channel utilization.

the performance of our approach in this subsection.

We conduct four sets of experiments. For all experiments, we use the same settings except changing the throughput of the AP. For each set, we test our detection algorithm 10 times. In the first set, we restrict the AP with idle workload, that is either the uplink or downlink throughput is smaller than 10kps. For the second set, to create 1Mbps throughput for both uplink and downlink of the AP, a wireless station is used to transmit UDP packets at constant bit rate to another wireless station through the AP. Similarly, 2Mbps and 5Mbps throughput are generated for the third and fourth set separately.

Fig. 2.15 shows the comparison of four kinds of situations. In the figure, the dark shaded bar describes the number of correct detections out of 10 times, and the light shaded bar indicates the number of wrong detections. We find our algorithm works well when the AP's workload is lower than 5Mbps. After that, the performance decreases.

Note that we only consider the case of an AP being falsely identified as a rogue AP due to the workload. We ignore the case that a rogue AP may suffer heavy workload, since that will help us to detect the rogue AP. Therefore, we assume a

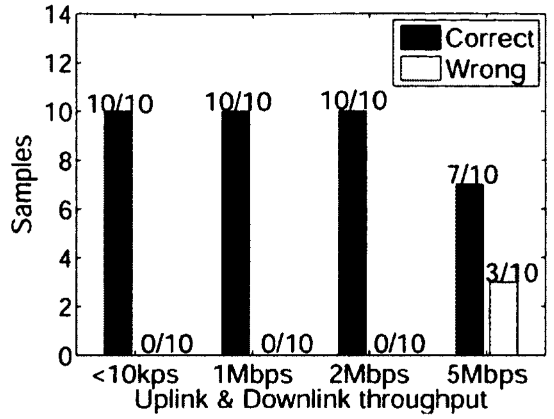


Figure 2.15: Histogram of the accuracy of our detection algorithm under different AP's workloads. The "Correct" means a legitimate AP is correctly identified, and the "Wrong" indicates falsely detecting an AP as a rogue AP.

rogue AP will never have heavy workload.

2.6.5 Number of samples

Previously, we found that our algorithm does not work well when the wireless traffic is saturated. Here, we are curious whether increasing n could improve the performance. In our experiments, we increase the original value of n from 100 to 300. Again, we test a legitimate AP and rogue AP separately under different channel utilization. Both tests are repeated 10 times. The detection accuracy is the ratio of the number of the tests in which the AP is correctly identified over the total number. Fig. 2.16 illustrates the accuracy of our algorithm against the channel utilization.

We find that using $n = 100$ is enough to achieve 100 percent of detection accuracy in low traffic situation. However, the accuracy falls to 65% as the channel becomes increasingly saturated. When setting $n = 300$, we are able to obtain 80% as the channel saturation increases.

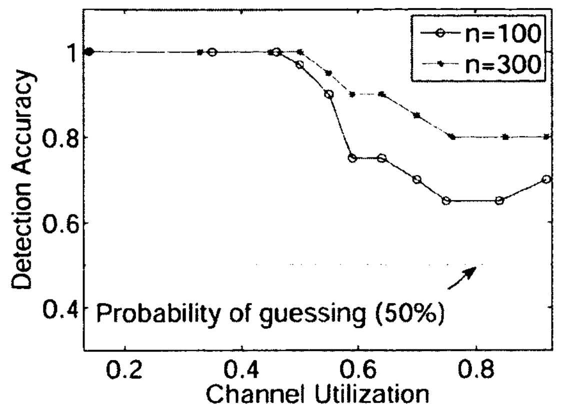


Figure 2.16: Detection accuracy against channel utilization

2.6.6 Testing time

To evaluate the efficiency of our detection algorithm, we use testing time, which is approximately the product of the mean value of RTT and the sample size. We do not consider other factors such as the time needed to associate to an AP or to obtain an IP address because those factors are dependent on specific AP configuration and may vary widely from one AP to another. Clearly, heavy traffic and large number of samples will lead to long testing time. Table 2.8 shows the time for testing a rogue AP under three traffic conditions. As we see, our algorithm requires less than one second under lightly-loaded traffic condition, and tens of seconds for heavy traffic condition.

	Detection time (second)			
	Idle	Half saturated	Saturated	
$n = 100$	0.2	0.9	11.3	AP
	0.3	1.1	17.2	RAP
$n = 300$	0.5	2.7	39.1	AP
	0.8	3.2	50.7	RAP

Table 2.8: Testing time under different network traffic conditions

2.7 Conclusion

The ease of setting up a successful rogue AP makes this form of wireless attack a serious security problem. While existing techniques can alleviate this threat, they nonetheless require active participation on the part of the network administrator. In this chapter, we present a practical, timing-based scheme for the end user to avoid connecting to rogue APs. This is done without any assistance from the network administrator.

3 Vehicular Rogue AP Detection

To deeply study the rogue AP problem, we consider another interesting and practical type of rogue AP which is set up in a moving vehicle for Drive-thru Internet. Unlike the rogue APs considered in the previous chapter, vehicular rogue APs are always moving and therefore more difficult to be detected. This chapter presents a novel scheme to prevent users from connecting to vehicular rogue APs.

3.1 Introduction

The *Drive-thru Internet* [7] has been of special interest in wireless communication research area for several years. The goal of Drive-thru Internet is to provide seamless Internet access to mobile users in moving vehicles by exploiting IEEE 802.11 technology [28,64]. Fig. 3.1 illustrates a typical Drive-thru Internet scenario, where the IEEE 802.11-based access points (APs) are deployed along the roads -- within the city or on a freeway. Mobile users in vehicles (i.e., vehicular clients) connect to these APs (so called *roadside APs*) for the Internet access. Recently, many city-wide Wi-Fi infrastructure for Drive-thru Internet has already been deployed in the real world. For example, Google provides a free wireless Internet service to the city of Mountain View [8].

Due to the ubiquitous deployment of APs, the problem of rogue APs has emerged as a well recognized security issue. A rogue AP is a malicious AP that pretends to be a legitimate AP to induce users to connect. Once an innocent user has associ-

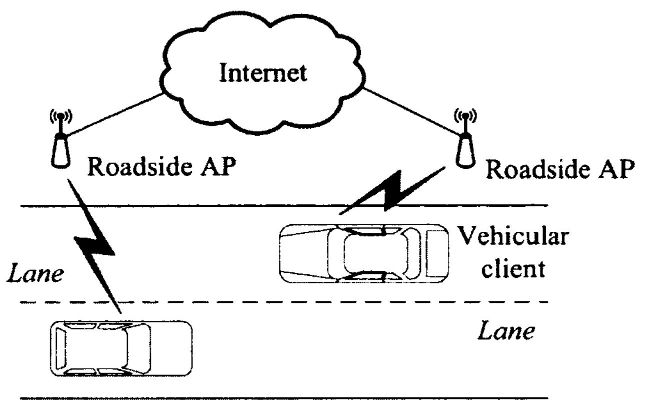


Figure 3.1: Illustration of a typical Drive-thru Internet scenario

ated to a rogue AP, the adversary can then launch various attacks via manipulating user's packets. For example, the adversary can launch phishing attacks to redirect user's web page requests to fake ones, seeking to steal user's secrets such as bank account numbers and passwords.

Rogue APs in vehicular networks can be broadly classified into two categories: *stationary* and *mobile*. In the first category, a stationary rogue AP is set up at a fixed place, such as in a building facing a busy road. Due to the mobility of vehicular clients, these types of rogue APs are unlikely to keep a long connection with clients. As a result, the time window for adversaries to steal users' secrets is short and the damage is restricted. In addition, a stationary rogue AP usually keeps active for a long time in a place. It is relatively not difficult for authorities to detect such a rogue AP. Previous works [17, 20, 24, 39, 57, 76, 83--86, 93] have already proposed several methods to detect stationary rogue APs. However, there is little work on how to defend against a mobile rogue AP or a vehicular rogue AP, where a malicious rogue AP is set up in a moving vehicle. Since a mobile rogue AP could follow traffic on a road, such an AP is able to maintain a long connection with users. Thus, this type of mobile rogue APs are more dangerous because they have more time to launch various attacks.

In this chapter, we consider the problem of detecting vehicular rogue APs from user's perspective. It is challenging because the duration for a vehicular client connected to an AP is short, so that the time left for the detection is restricted. According to IEEE 802.11 standards, when the signal strength of a connected AP is less than a threshold, the client will perform a handoff [36] and re-associate to another AP with the strongest signal strength nearby. Accordingly, it is meaningless to determine whether or not an AP is rogue while such an AP is out of the client's reach. Another challenge is that the information obtained by clients has to go through APs. To evade detection, a rogue AP can impersonate a legitimate AP by providing fake information such as MAC address, BSSID and other configurations. There is no trusted authorities that can be used to validate this information.

Considering the above challenges, we propose a practical detection scheme that prevents users from connecting to vehicular rogue APs. Our solution is compatible with IEEE 802.11 standards and imposes little modification to the AP side. The only change to existing APs is to add their GPS information in each beacon they broadcast. Based on that, clients are able to measure the received signal strength (RSS), coupled with several test packets with controlled transmission (tx) power and data rate to detect rogue APs. To the best of our knowledge, we are the first to consider the vehicular rogue AP problem and propose a practical user-side detection scheme. Our main contributions are listed as follows:

1. We are the first to study the vehicular rogue AP problem. The vehicular rogue APs considered in this chapter are set up in moving vehicles by malicious attackers. They are powerful and capable of forging anything to escape from detection.

2. We propose a practical scheme to defend against vehicular rogue APs. Our approach is a pure user-centric method that can be performed by end users without any help from network administrators. In addition, our approach is compatible with 802.11 standards.

3. We implement our scheme using commercial off-the-shelf devices including APs, wireless cards, antennas, and GPS modules. In addition, we show our scheme is effective in real world environment through extensive experiments on realistic road conditions.

The rest of the chapter is organized as follows. Section 3.2 discusses the related work. Section 3.3 describes the adversary model. Our detection algorithms are detailed in Section 3.4. The implementation and evaluation are presented in Section 3.5. Last, we discuss the limitation of our solution in Section 3.6 and conclude in Section 3.7.

3.2 Related Work

The threat of rogue APs has attracted the attention of both industrial and academic researchers. Previous research has been mainly focused on detecting static rogue APs in enterprise or hotspot scenarios. Existing schemes can be broadly classified into to three categories.

The first category relies on sniffers to monitor wireless traffic. These sniffers usually scan spectrum to examine the 2.4 and 5GHz frequencies. Once detecting any traffic from unauthorized APs, they will alert the administrator. This approach usually demands well controlled infrastructure such as enterprise networks, where the administrator can easily deploy sniffers and cut off the access of rogue APs to Internet. Some commercial products [1--3] have been developed following this technique. In academic community, an architecture for diagnosing various faults in WLAN including rogue APs, is presented in [17], where multiple APs and mobile clients installing a special diagnostic software cooperate to perform RF monitoring. In [20], another monitoring infrastructure called DAIR is proposed, where USB wireless adapters are attached to desktop machines for capturing more comprehensive traffic to reduce false detection rate. Different from this type of solution, our defending scheme does not reply on sniffers. That is because a small amount of sniffers may not catch vehicular rogue APs well, but extensive deployment of sniffers is impractical.

The technique used in the second category is leveraging fingerprints to identify rogue APs. Since an advanced adversary can easily spoof a rogue AP's MAC address, SSID, vendor name, and configuration to escape from the detection, the previous work often adopted the fingerprints that cannot be easily forged. For example, the work by [46] calculated every AP's clock skew by collecting their beacons and probe responses. Since clock skew is difficult to forge , any AP with unknown clock skew is identified as a rogue AP. In addition to clock skew, RSS values [75], radio frequency variations [26] are also used. However, a major drawback of this type of schemes is that the AP validation requires to access a database containing the fingerprints of all legitimate APs. This database may not be available for end users before they connect to the AP. Our solution differs from such schemes in that we do not assume the clients know the fingerprints of legitimate APs in advance. Therefore, our detection scheme can apply to not only network administrators but also end users who use the wireless network for the first time.

The last category exploits the features of wireless traffic to detect the presence of rogue APs [39,57,86,93]. In [39], a practical timing based scheme is proposed. The method employs the round trip time between user and local DNS server to detect rogue APs without assistance from network administrators. [84] utilizes the immediate switch connecting rogue APs to measure round trip time of TCP traffic. Other work by [24, 76] uses the spacing between packets to distinguish wireless networks from wired networks. In [86], inter-arrival time of ACK-pair is used to detect rogue APs. In [93], wired verifier and wireless sniffers are deployed at the same time to detect layer-3 rogue APs. Since we consider a new type of rogue AP, it is unclear whether previous solutions could work in vehicular networks. Although some schemes might work, it may spend much time on analyzing the network traffic. Our solution is more efficient and user-centric.

3.3 Adversary model

Vehicular rogue APs considered in this chapter are set up in moving vehicles by malicious attackers. The goal of the rogue AP is to induce clients to connect. If any client associates to a vehicular rogue AP, the adversaries succeed. Several assumptions are described as follows.

(1) We assume the vehicular rogue APs and vehicular clients stay on the same lane of a road and follow traffic. In other words, the relative distance between a rogue AP and a client will not change too much. The vehicular rogue APs on the opposite lane are not considered, because the time window for them to launch further attacks is extremely limited. Even though some clients may associate to the rogue APs on the other lane by accident, they will quickly move out of the range.

(2) Previous work [39] assumes that rogue APs have to connect to existing legitimate APs to access Internet, so they suffer from multi-hop wireless transmission and are slower than legitimate APs. Here we do not hold this assumption, as well as any other assumptions about the back-end infrastructure for vehicular rogue APs to access Internet.

(3) We assume that rogue APs are able to transmit any packet with arbitrary tx power and inject packets with any content. For example, the adversary can generate fake re-associate frames to force clients to re-select APs immediately, and provide whatever fake information to clients.

(4) We assume the adversary cannot modify the firmware of wireless cards. For example, the rogue AP cannot control Ack frames. Although Ack frames can be

generated by software, they cannot be sent back to clients within pre-determined Ack timeout. It is mentioned that the time interval between data and Ack is a SIFS (10 μs in 802.11g [82]). Software (not firmware) Ack cannot be prepared within such a short interval [62].

(5) We do not consider the Drive-thru Internet is capable of using RADIUS-based 802.1X authentication of users, since it requires each vehicle to have the correct key to access the network. While this may be possible, for example, every car when registered with the DMV is issued the appropriate credentials, it is unclear how well this will work in practice. Thus, we only consider the open 802.11 network where any car can connect.

Based on these assumptions, the adversary can launch two types of attacks.

Basic attack. In the basic attack, vehicular rogue APs broadcast beacons with the maximum tx power. The maximum power will lead to the strongest signal strength with which the vehicular rogue APs have most probability to attract users to connect. The advantage of the basic attack is its easy setup. Without any complicated configuration, the basic attack can be launched anytime and anywhere.

Advanced attack. The major difference between the advanced attack and the basic attack is that the former attack needs background preparation before creating a vehicular rogue AP. For example, the adversary could first drive along the road as a client to profile RSS values from existing roadside APs, and then tune tx power to make the signal strength received by nearby clients similar to the profiled values. By doing so, RSS-based solution cannot work well, because the RSS values measured by the clients appear like "real". Compared to the basic attack, the advanced attack is more time-consuming, but is more difficult to be detected.

3.4 Our Solution

The algorithms for detecting basic attacks and advanced attacks are presented separately. In both algorithms, each AP needs to broadcast its physical location such as GPS coordinates. Since the IEEE 802.11 standards allow adding arbitrary information elements with variable-length in beacons, the GPS information can be easily included in beacons which are broadcasted by every AP periodically. To determine AP's location, the network administrator can resort to an external GPS module and measure it offline. Upon receiving a beacon, clients know the AP's location. If any AP refuses to expose its GPS location, the client will never connect to that AP for security consideration. For a legitimate AP, it always reports its actual GPS coordinates which indicate a fixed location alongside the road. However, a vehicular rogue AP is afraid of reporting its true GPS location, because that location indicates such an AP is always moving on the road. A user can easily detect it and avoid connecting to that AP. Hence the vehicular rogue AP has to forge GPS location to escape from the simple detection. The problem of detecting rogue APs is then converted to the problem of verifying whether an AP lies on its GPS location. Any AP who lies on its location is deemed as a rogue AP; otherwise, it is a legitimate AP.

3.4.1 Defending against basic attacks

First, we present the algorithm to defend against basic attacks. The basic idea of this algorithm is to leverage RSS on the client side to verify whether the rogue AP lies on its location. Since the rogue AP is not physically at the reported location, the RSS values should be able to tell the difference especially when the car is always moving. The intuition is shown in Fig. 3.2. Suppose a vehicular client moves on a road and receives m beacons at different places. The RSS value of each beacon

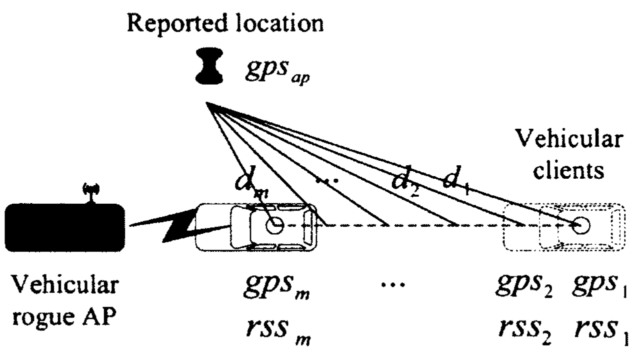


Figure 3.2: Intuition of defending against basic attacks, where a vehicular client needs to measure RSS and its own GPS location at multiple places

is denoted as rss_i where $i \in [1, \cdots, m]$, and each location gps_i indicates the exact location where the beacon is received. Thus, based on AP's reported location gps_{ap} and its own gps_i, the client can continuously compute the distance between the AP and itself (i.e., $d_1, d_2, \cdots, d_m$). On the other hand, the distance can also be inferred from RSS values. If the distances computed from GPS coordinates significantly differ from those inferred from RSS values, that AP is highly likely to be a vehicular rogue AP.

How to infer the distance from RSS? We adopt a widely used log-distance propagation model to characterize the relationship between distance and RSS. In this model, the received signal power decreases logarithmically with distance and it is expressed as

$$P_r(d) = \frac{c \cdot P_t}{d^{\gamma}}, \tag{3.1}$$

where P_t is referred to as transmit power of the sender, $P_r(d)$ is the received signal strength at a distance of d, and c is a correction constant which captures the effects of transmit frequency, antenna gains of both sender and receiver, as well as other factors in the environment. The *path loss exponent* γ determines the rate of attenuation when the signal propagating through the space, which is dependent on the propagation environment. A larger γ means the environment is lossy and will cause the fall of RSS faster with distance.

In the scale of decibel (dB), Eq. 3.1 can be rewritten as

$$P_r(d) = P_t + c - 10\gamma \log_{10}(d) + X_\delta \tag{3.2}$$

which depicts a linear relationship between RSS and the logarithmical distance. The newly introduced X_δ is a random variable with zero mean, which reflects the attenuation caused by flat fading [31]. Such a log-distance model has been verified to predict RSS well in various locations including outdoor scenario by previous research [65] and our real world experiments. However, the model parameters (i.e., c and γ) should be adjusted according to specific outdoor space. As mentioned in [41,65], γ normally ranges from 2 to 6, where 2 is for propagation in free space and 6 is for heavily lossy environment.

Overview of algorithm. The overview of the algorithm is described in Algorithm 3. For each AP that a client tries to associate to, this algorithm is performed to test whether such an AP is a rogue AP. The inputs of the algorithm are gps_{ap}, time series data rss_i and gps_i where $i \in [1, 2, \cdots, m]$. Namely gps_{ap} denotes the reported location of the tested AP, rss_i is the received signal strength of the ith beacon, and gps_i is the location of the client where the ith beacon is received. The parameter m determines the number of samples demanded to perform the algorithm. Now let us assume all inputs are available. Later, we will elaborate how to obtain them. The output of the algorithm is either *true* or *false*, which means the tested AP is a rogue AP or not respectively. Initially, the client computes the distance d_i between the reported location of the AP and itself (from line 1 to line 3). Next, rss_i and d_i are fit into Eq. 3.2 to estimate the parameter γ using least square method. If the value of γ is within the normal range from 2 to 6, the algorithm terminates and returns false otherwise true.

Fig. 3.3 illustrates an example of the least square fit of the samples collected

Algorithm 3 Detecting Basic Rogue APs

```
Input: $gps_{ap}$, $rss_i$ and $gps_i$ where $i \in [1, 2, \cdots, m]$
Output: *true* or *false*
1: for $k = 1$ to $m$ do
2:     $d_i = \|gps_{ap} - gps_i\|$
3: end for
4: Use least square fit of $< rss_i, d_i >$ to estimate $\gamma$
5: if $\gamma$ falls into the range from 2 to 6 then
6:     Return false
7: else
8:     Return true
9: end if
```

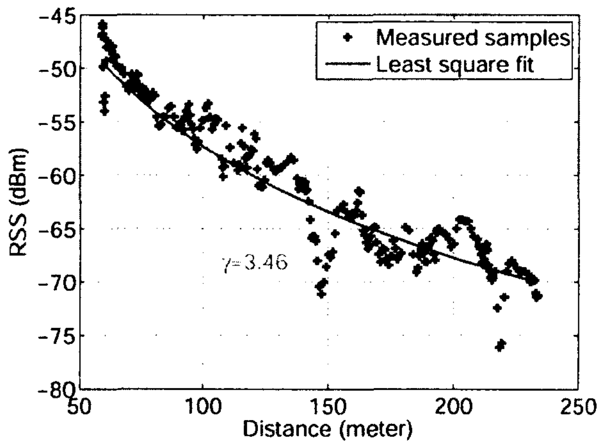


Figure 3.3: Illustration of the least square fit of the measured samples collected from a real roadside AP

from a real roadside AP. As we see, the estimated parameter γ indeed falls into the normal range from 2 to 6. We also tested many other APs using different devices and in varying environment. They all have similar results, but most vehicular rogue APs yield out-of-range results. The results are presented in Section 3.5.

How to obtain GPS and RSS? When receiving a beacon, a client can measure its RSS value. However, the frequency of GPS update in practice is much slower than that of receiving a beacon. For example, most commercial GPS modules update GPS coordinates less than 1 Hz, whereas a client may receive about 10 beacons within 1 second. It then raises a problem of how to assign a location for each beacon. To address this problem, we assume that the client did not change the speed during the interval of GPS update. This is true, because the interval is very small so that the speed diversity can be ignored. Next we apply interpolation on GPS data. Suppose two GPS updates occur at timestamps t_i and t_j, and the coordinates change from $gps(t_i)$ to $gps(t_j)$. Between them, the client receives several beacons with rss_k where $i \leq k \leq j$. To estimate $gps(t_k)$, we have

$$gps(t_k) = gps(t_i) + (gps(t_j) - gps(t_i)) \cdot \frac{t_k - t_i}{t_j - t_i}.$$

How to deal with RSS noise? Due to many reasons such as dynamic environment interference, the measured RSS may suffer from some extreme values. With these abnormal values the model parameters may be estimated inaccurately. Therefore, we have to filter out them before applying the least square fit. The process is simple as follows: the client checks consecutive three RSS values. If the difference between the median value and the average of its prior and following values exceeds a threshold τ, the median value is replaced by the average value. Heuristically, we set τ to 5 dBm.

How to determine the number of samples? The number of samples m affects the time duration and the accuracy of the detection. A larger m typically leads to a more accurate result but costs longer time to finish. To determine m, we propose a method that can adaptively involve more samples until the estimated parameter γ becomes stable. In the approach, once an AP is first discovered, the client keeps measuring RSS in background. Every time duration Δt (second), all the accumulated RSS values for that AP are fitted into Eq. 3.2 to estimate γ. If γ does not change within a threshold θ for continuous two Δt, γ is deemed to be stable and we stop to involve more samples. This method works well in practice by setting $\Delta t = 1$ and $\theta = 0.5$. The experiment results are presented in Section 3.5.

3.4.2 Defending against advanced attacks

As mentioned in Section 3.3, by profiling in advance, a sophisticated rogue AP is able to tune tx power to mimic the RSS trend of a roadside AP. In this case, the detection cannot merely rely on RSS values because they appear like "real". Hence, our algorithm for basic attacks cannot work. It is much more challenging than defending against basic attacks, because the adversary is more sophisticated. In this section, we present a novel user-side algorithm for defending against advanced attacks. This algorithm demand clients to send probe requests to the AP and perform a series of lie-detection tests. Note that this algorithm is complementary to the previous algorithm. They can be combined together to make the detection more accurate.

Background. Before presenting the detail, let us first introduce some background knowledge. In wireless communication, the transmission distance that can be achieved between two wireless devices is influenced by the tx power of the sender and the rx sensitivity of the receiver. Out of this distance, the receiver cannot receive packets correctly. As a sender, there are two ways to adjust the transmission distance in purpose. Tweaking tx power is one of them. Fig. 3.4 shows RSS values versus various tx powers at distance of one meter between a sender and a receiver in our experiments. As we see, the average RSS values decrease linearly with tx power. That means large tx power leads to long communication distance.

Using rx sensitivity is another way to change the transmission distance. With greater rx sensitivity, the device is able to receive weaker signals, thus owning longer transmission distance. In particular, rx sensitivity is influenced by data rate which is controlled by the sender. Different data rate achieved by different modulation schemes requires different rx sensitivity. When RSS value is less than rx

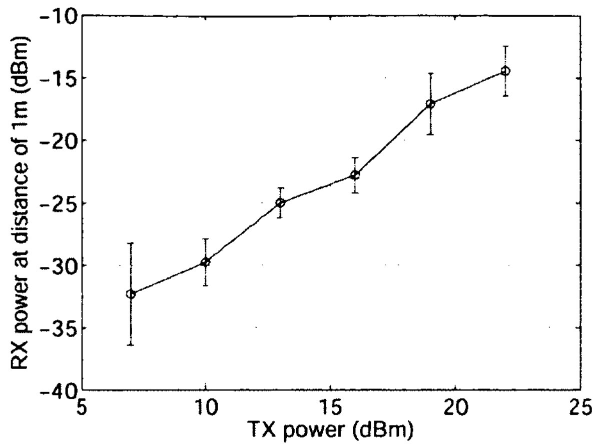


Figure 3.4: Average RSS values versus varying tx powers

Rates (Mbps)	Standard	Modulation	Rx sensitivity (dBm)
6	802.11a/g	BPSK/OFDM	-82
9	802.11a/g	BPSK/OFDM	-81
12	802.11a/g	QPSK/OFDM	-79
18	802.11a/g	QPSK/OFDM	-77
24	802.11a/g	QAM-16/OFDM	-74
36	802.11a/g	QAM-16/OFDM	-70
48	802.11a/g	QAM-64/OFDM	-66
54	802.11a/g	QAM-64/OFDM	-65

Table 3.1: Modulations for 802.11a/g

sensitivity, the packet cannot be demodulated correctly. Table 3.1 presents an example of mapping between data rate (modulation) and rx sensitivity for each data rate supported in 802.11a/g networks. It is seen that a fast data rate typically requires a large rx sensitivity, thus leading to a short communication distance.

Intuition. Given a certain tx power and data rate, the communication range between the client and the AP is determined. If the reported location yields a significant departure from that range, there should be something wrong. For example, an AP claims far away from the client, but it can receive packets with very low tx powers and high data rates. Such an AP is likely to be a rogue AP. The client herself could perform this test according to the reported location of the AP. However, the problem is that different AP may have different rx sensitivity for each data rate, which may affect the correctness of the test. For instance, the succuss in receiving a packet with low tx power and high data rate may be due to a powerful antenna rather than a short distance. To cope with this problem, our algorithm choose a different strategy that the client will test each AP with a vector of power-and-rate combinations in the order from the low-power/high-rate end to the high-power/low-rate end. In this vector, the AP could receive the packets at the beginning but fail after a certain combination. Such a combination is labeled as the boundary of the vector, which reflects the communication distance. After testing several times, if the client finds that the reported distance changes a lot but the boundary rarely changes, then such an AP is deemed as a rogue AP. The rationale behind this idea is that the relative distance between a legitimate roadside AP and a client changes when the client is moving on the road, but the distance between a mobile rogue AP and the client never changes so much.

Overview of the algorithm. Algorithm 4 describes the detail. The procedure *send_probe_requests* is used to send probe requests with varying tx powers and data rates to test an AP. Instead of exhausting all the combinations, we only picks a subset to reduce the finish time. Suppose the maximum tx power of a client is $pwr_{max} = 27dBm$, then the power set is $\{7dBm, 17dBm, 27dBm\}$. The selected rate set is always limited to $\{54Mbps, 48Mbps, 36Mbps, 24Mbps\}$, since it can separate the 10dBm difference well. The vector of combinations is as follow:

$$\{7dBm/54Mbps, 7dBm/48Mbps, 7dBm/24Mbps, \cdots, 27dBm/24Mbps\}.$$

It should be noticed that according to our experiments the current ordering of the vector strictly follows the communication distance from short to long . For each combination, the client sends several probe requests to the AP. If more than half of packets got Ack back, such an combination is deemed as *receivable* else *non-*

Algorithm 4 Detecting Advanced Rogue APs

```
procedure: send_probe_requests
$pwrset \leftarrow \{pwr_{max}\%10, pwr_{max}\%10 + 10, \cdots, pwr_{max}\}$
$rateset \leftarrow \{54, 48, 36, 24\}$
for $i = 1$ to $size(pwrset) * size(rateset)$ do
    Set tx power to $pwrset[ceil(\frac{i}{size(rateset)})]$
    Set data rate to $rateset[i\%size(rateset)]$
    Send $n$ probe requests to AP with tx power and data rate above. If more than
    half receive Ack back, then $V[i] = 1$; Otherwise, $V[i] = 0$
end for
1: if AP is first discovered (weak RSS) then
2:     Call send_probe to obtain a vector $V$
3: end if
4: if client intends to associate to that AP (strong RSS) then
5:     Call send_probe to obtain another vector $V'$
6: end if
7: Find the boundary in both $V$ and $V'$ such that the ratio of 1 to the number of
   elements before the boundary and the ratio of 0 to the number of elements after
   the boundary are maximized.
8: if $boundary(V) - boundary(V') < threshold$ then
9:     Such an AP is a rogue AP
10: end if
```

receivable, denoted by 1 or 0 separately. Based on this vector, the boundary is computed such that the ratio of 1 to the number of elements before the boundary and the ratio of 0 to the number of elements after the boundary are maximized. In the algorithm, the client calls this procedure twice: once at the time when first discovering the AP (weak RSS), and the other time when trying to associate to that AP (strong RSS). By comparing the boundary of two vectors, if the difference does not exceed a threshold, the client will never connect to that AP. The reason is that the physical distance has changed a lot so the boundary difference should reflect this change. If not, such an AP is deemed as a rogue AP. In this chapter, we heuristically set this threshold to 4 based on our experiments.

3.5 Evaluation

In this section, we present our experimental setup, the methodology and experimental results, which attempt to answer the following questions: 1) What is the performance of both basic algorithm and advanced algorithm working in practice? 2) What is the time cost of determining whether or not an AP is a rogue AP? 3) How does the speed of a vehicle affect the performance?

3.5.1 Experimental setup and methodology

The devices used in our experiments are comprised of a roadside AP, a vehicular rogue AP, and a vehicular client.

Roadside AP. A commercial outdoor AP (Deliberant CPE 2-12) was configured as a roadside AP. The specification of this model can be found in [6]. The AP was mounted on top of a tripod that is 2m high (see the left part of Fig. 3.5). When deploying the AP alongside the road, we used a GPS receiver (GlobalSat BU-353) to measure its physical location. To enable broadcasting the GPS information via beacons, we loaded the AP with OpenWrt [15] firmware and a modified Wi-Fi driver. The extra content in each beacon has 18 bytes including 1 byte element ID, 1 byte length, 8 bytes latitude, and 8 bytes longitude.

Vehicular rogue AP. A laptop connected with an external omni-antenna and a GPS receiver (see the right part of Fig. 3.5) mounted on the roof of a car was configured as a vehicular rogue AP. The laptop was running a 2.6.27-generic Linux kernel with madwifi driver (svn r4128). Similar to the roadside AP, we modified the madwifi driver to support GPS broadcast. We did not set up the Internet access for all APs, since it does not affect the performance of our algorithms. In basic attacks, we fixed the tx power by executing command `iwconfig txpower [value]` with the maximum power value. In advanced attacks, we tried to automatically adjust tx

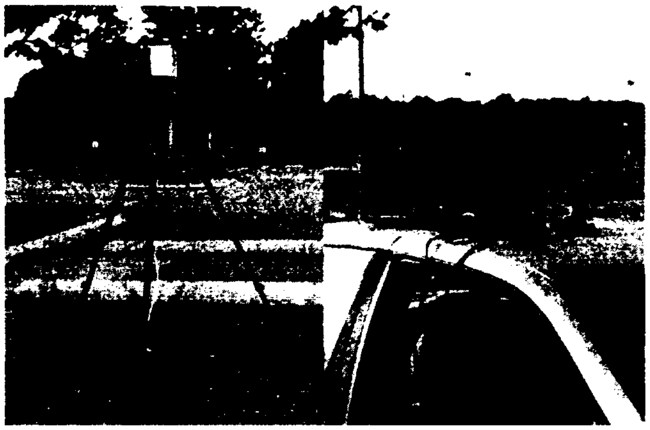


Figure 3.5: Experimental setups of a roadside AP (left) and a vehicular rogue AP (right)

power to mimic the real trend of RSS values, but eventually we found that it was really difficulty to make it work in practice. Most of time, the advanced rogue AP could be detected by our basic algorithm. To ease evaluation, we optimistically assume the rogue AP can escape from our basic algorithm and evaluate our advanced algorithm without changing the tx power of the AP. This is reasonable because our algorithm does not rely on any configuration of APs.

Vehicular client. The vehicular client used the same hardware as the vehicular rogue AP. The Wi-Fi interface of the client was set to monitor mode which could capture all the packets in air. Injecting and receiving packets were achieved by libpcap. The control of per-packet tx power and data rate was done by radiotap header. In Linux, the IEEE 802.11 MAC layer allows arbitrary injected packet composed in the following format:

[radiotap header] + [ieee80211 header] + [payload].

`IEEE80211_RADIOTAP_RATE` and `IEEE80211_RADIOTAP_DMB_TX_POWER` in the radiotap header are used to control the data rate and the tx power of injected packets. Given different values, a packet can be transmitted with the desired power and data rate. Note that to control per-packet tx power, `hal_tpc` must be enabled while

Name	Notation
AP	Deliberant CPE 2-12 based on WDB-500 platform
Laptop	Lenovo T61 with 2.0GHz processor and 1G RAM
GPS	GlobalSat BU-353 USB GPS receiver
Wireless card	CB9-GP Cardbus 802.11a/b/g based on Atheros chipset
Antenna	7 dBi MA24-7N magnetic-mount omnidirectional

Table 3.2: Equipment description

loading the madwifi module. Table 3.2 summarizes all the equipment used in our experiments.

The experiments were conducted in a suburban area, where we could freely drive along the road and stop to collect measurements. In the experiments, a roadside AP was placed at a parking lot around 60m away from the road. Two cars configured to be a vehicular rogue AP and a vehicular client were driven along the road passing through the roadside AP. The roadside AP broadcasted its actual GPS location, and the rogue AP broadcasted a location close to the roadside AP. We took two sets of experiments to evaluate our vehicular rogue AP detection schemes. The first set of experiments was used to evaluate the performance of our basic algorithm, where the client passively listened beacons. The second set was to evaluate the advanced algorithm, where the client actively sent probe requests to the AP.

3.5.2 Experimental results

Basic attack evaluation. First, we tested if legitimate roadside APs could pass our basic algorithm. As an example, Fig. 3.6 illustrates the measured RSS values against the logarithmical distance in an experiment. The client started the algorithm when observing the first beacon from a roadside AP and terminated the algorithm when γ was stable. In total, the client collected 110 beacons within 11 seconds. The estimated γ was 3.31 eventually, which falls into the valid range from 2 to 6.

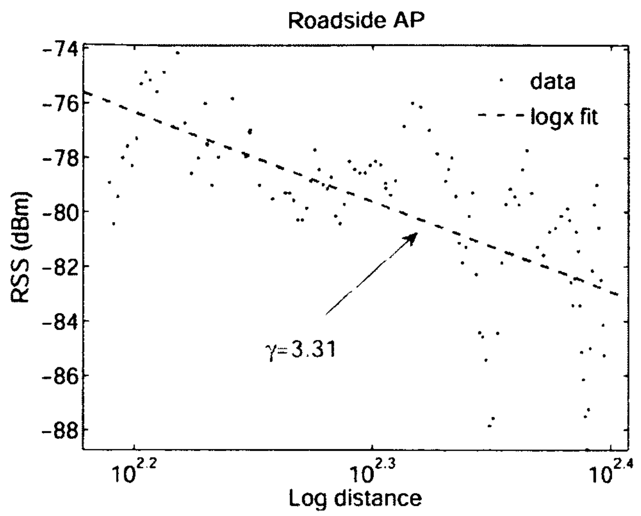


Figure 3.6: Result of our basic algorithm in the case of a legitimate AP

Therefore, the AP is labeled as a legitimate AP correctly. Although the finish time seems a little long, it should be noticed that this cost only occurs once when the client initially turns on the Wi-Fi and tries to find an AP to connect. After that, the client will wait for a certain period until the signal strength of current AP becomes weak. Only at that time, the client needs to find another AP for handoff. During the waiting period, the client should have collected enough RSS values from nearby APs to determine which APs are rogue APs. To investigate the robustness of our algorithm, we also conducted experiments in different environments. We observed that γ varied across environments but they all felt into the normal range.

Next, we tested the performance of the algorithm when the tested AP is a rogue AP. Fig. 3.7 shows the result. As we see, after collecting 90 beacons, γ was stable at 1. It is clear that the tested AP is a rogue AP.

Advanced attack evaluation. Fig. 3.8 and Fig. 3.9 depicts the experimental results of our advanced algorithm in respect to a legitimate roadside AP and a vehicular rogue AP. The figure only shows a part of the vector which contains the $0/1$ boundary. The first test occurred when the first beacon was received, and the second test was performed when the client tried to associate to that AP. It is seen

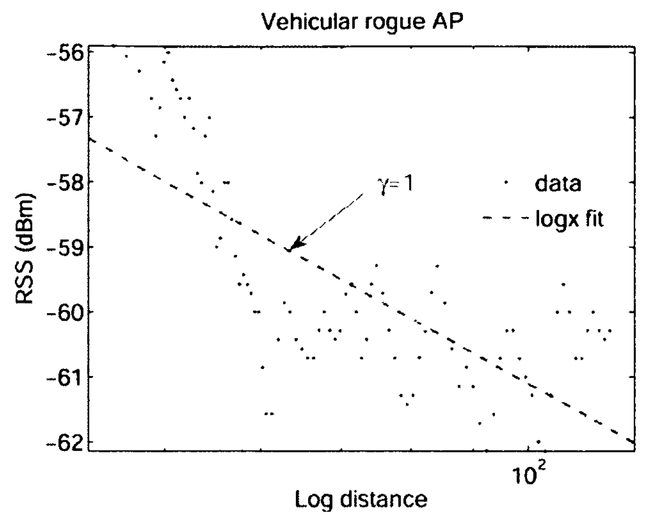


Figure 3.7: Results of our basic algorithm in the case of a vehicular rogue AP

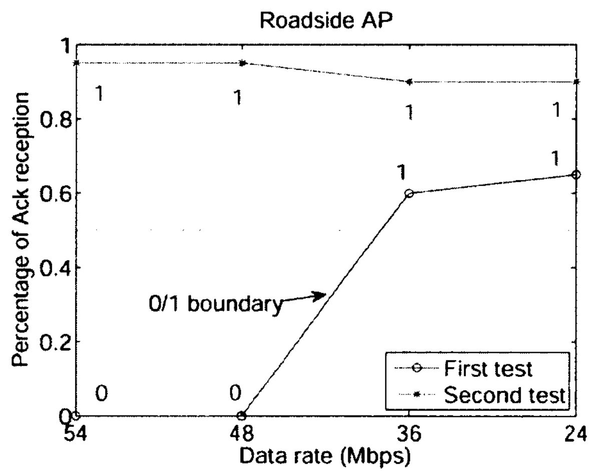


Figure 3.8: Results of our advanced algorithm in case of a roadside AP

that the boundary for a roadside AP changed significantly due to the mobility of the vehicle. By contrast, the boundary changed little when the tested AP is a rogue AP. We also evaluated the performance at different locations. We observed that our algorithm could achieve more than 90% accurate to label an AP correctly. Due to page limit, the results are not presented in this chapter.

Finish time versus vehicle speed. The finish time of our basic algorithm is determined by the duration when γ becomes stable. We have investigated the finish time and the difference of γ (the value when the algorithm is terminated to the ground truth) against different vehicle speeds. The finish time is measured in unit

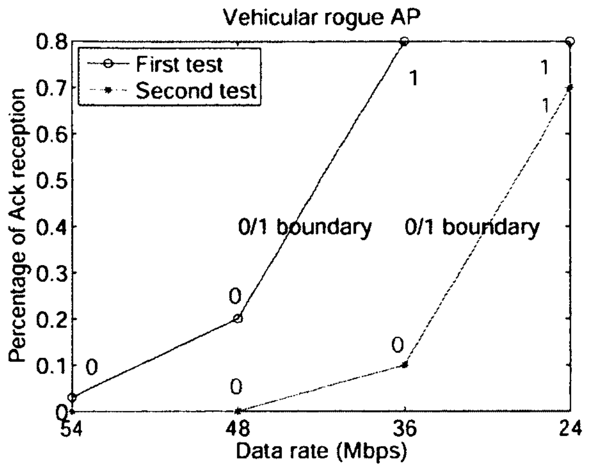


Figure 3.9: Results of our advanced algorithm in case of a vehicular rogue AP

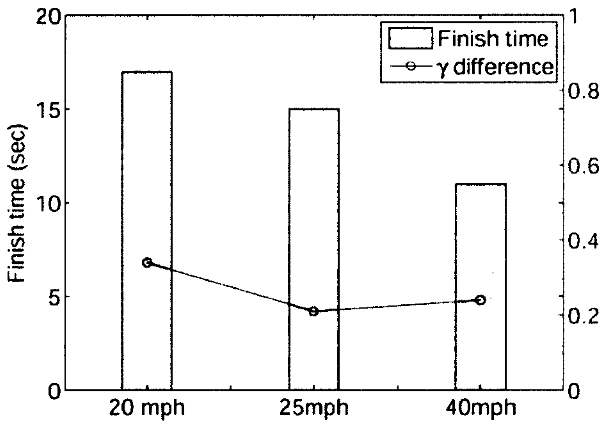


Figure 3.10: Finish time and γ difference versus the vehicle speed

of second, and the ground truth of γ is derived when all beacons (from entering to leaving the communication range of the AP) are used for least square fit. Fig. 3.10 presents the results. As we see, the faster the speed of vehicle, the quicker the algorithm can finish. Again, this cost only incurs when the Wi-Fi interface is initially turned on. After that the cost can be amortized by background scan. In addition, we find our algorithm can estimate γ accurately. The difference of γ is within 0.3.

3.6 Discussion

Our solutions make use of physical characteristic such as path loss and the percentage of acknowledged tx packets to determine the discrepancy between the rogue AP's actual location and its reported location. This makes our solutions vulnerable to the following factors.

One factor is that outdoor wireless condition is unpredictable. Although our solution relies on a well-known signal propagation models, it is inevitable that there will be instances where the actual conditions deviate from the models. When this happens, our solution is not able to detect the rogue AP successfully. We can mitigate by adopting a more accurate model in our solution. In addition, it is unclear how well the adversary can take advantage of this limitation since the adversary is unable to predict the channel conditions as well. Finally, we have observed that our solutions may not work as well in some locations where there are a lot of buildings. We will investigate more complex environments and refine our algorithms in our future work.

Another factor is that the wireless interference may have a negative effect on the packet reception. It is difficult for a sender to infer that an unsuccessful packet transmission is caused by either bad signal strength or interference. When this happens, our advanced algorithm may not detect the rogue AP well. However, the adversary cannot easily utilize this uncontrolled factor to increase the probability of escaping from the detection. In the future work, we will study how to reduce the detection errors caused by the interference.

3.7 Conclusion

Attacks from vehicular rogue APs have raised a serious security problem for Drive-thru Internet. In this chapter, we are the first to demonstrate the feasibility of this type of rogue APs and present a practical defending scheme to prevent users from connecting to vehicular rogue APs. We implement our approach on commercially available hardware and perform extensive real-world experiments that confirm the efficacy of our solutions.

4 Power Saving for Wi-Fi Tethering

The previous chapters discover the guidelines for improving mobile security. However, the concepts of security and energy are closely related with each other. A system or algorithm design for mobile computing that considers only security but not energy is unlikely to work well in practice. If an enhancement is at the expense of the energy consumption, it will not be accepted by battery-powered mobile devices. In this chapter we study the problem of how to save power without compromising the user experience. Our power-efficient system called DozyAP can significantly reduce the power consumption of mobile devices working as softAPs in Wi-Fi tethering, while retaining a good user experience.

4.1 Introduction

Wi-Fi tethering, also known as a "mobile hotspot", means sharing the Internet connection (e.g., a 3G connection) of an Internet-capable mobile phone with other devices over Wi-Fi. As shown in Figure 4.1, a Wi-Fi tethering mobile phone acts as a mobile software access point (softAP). Other devices such as laptops, tablet PCs and other mobile phones can connect to the mobile softAP through their Wi-Fi interfaces. The mobile softAP routes the data packets between its 3G interface and its Wi-Fi interface. Consequently, all the devices connected to the mobile softAP are able to access the Internet.

Wi-Fi tethering is highly desired. Main-streaming smartphones including iPhones

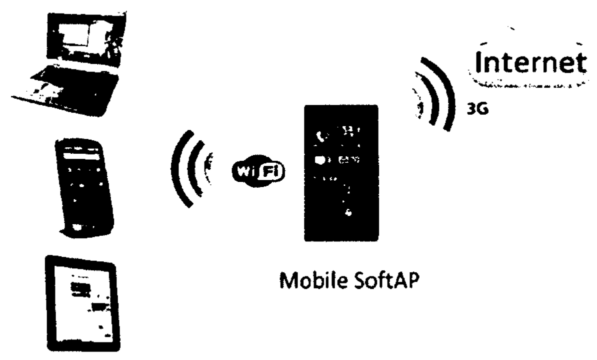


Figure 4.1: Illustration of a typical setting of Wi-Fi tethering

(iOS 4.3+), Android phones (Android 2.2+) and Windows phones (Windows Phone 7.5+) all provide built-in support of Wi-Fi tethering. There were also many third-party Wi-Fi tethering tools with customized features in App markets. We believe there are two main reasons why Wi-Fi tethering is so desirable. First, cellular data networks provide ubiquitous Internet access over the world but the coverage of Wi-Fi networks is much limited. Second, it is common for people to own multiple mobile devices but likely they do not have a dedicated cellular data plan for every device. As a result, it demands to share a data plan among multiple devices, e.g., sharing the 3G connection of an iPhone with a Wi-Fi only iPad. Wi-Fi tethering provides a convenient way to do this.

However, Wi-Fi tethering significantly burdens smartphone's battery. When enabling tethering, the Wi-Fi interface always stays in high power state and never sleeps even when there is no data traffic going on. This increases the power consumption by one order of magnitude and reduces the battery life from days to hours (more details in Section 4.2.1). To save power, Windows Phone automatically turns off Wi-Fi tethering if the network is inactive for a time threshold of several minutes. However, this method has two drawbacks. First, the Wi-Fi interface still operates in a high power state for the idle intervals less than the threshold, leading to waste of energy. Second, it harms usability. If a user does not generate any traffic for a time period longer than the threshold (e.g., while reading a long news article) and then starts to use the network again (e.g., by clicking another news link), the user will have to go back to the smartphone and manually re-enable Wi-Fi tethering, which results in poor user experience.

The IEEE 802.11 standard defines the power saving mechanism for wireless stations in client mode, ad hoc mode, but not in AP mode. That is because traditional APs are externally powered by cables, so that power saving is not an crucial issue for those APs. However, old wisdom does not work for the battery-powered smartphones operating in Wi-Fi tethering. Hence, it is time to think about how to save power for smartphones working as softAPs.

Recently, Wi-Fi Direct specification introduces a power saving protocol for Wi-Fi Direct devices acting as APs. The protocol operates in the media access control (MAC) Layer and allows APs to notify clients with newly defined messages when they are going to sleep. However, existing 802.11 devices including most smartphones cannot benefit from the new feature supplied by Wi-Fi Direct. Future mobile devices may have a chipset that can support Wi-Fi Direct. However, due to the lack of the programmability of Wi-Fi chipsets in Wi-Fi industry, a device vendor still may find it difficult to implement its own tethering solution independent of the one delivered by the chipset vendor. In this chapter, we propose a new approach for device vendors and software developers to implement a power-efficient Wi-Fi tethering solution without underlying support. To demonstrate its efficacy, we design *DozyAP*, a system to reduce power consumption of Wi-Fi tethering on smartphones while still retaining a good user experience.

The key idea of DozyAP is to put the Wi-Fi interface of a softAP into sleep to save power. We measured the traffic pattern of various online applications used in Wi-Fi tethering. We find that the Wi-Fi network is idle for a large portion of total application time (more details in Section 4.2.2), which means the AP could sleep during this idle time. Furthermore, we know that the cellular interface is typically slower than Wi-Fi interface. Thus, the Wi-Fi interface of a softAP could sleep while waiting for the data transmission through the cellular network. All of these indicates there are many opportunities to reduce softAP's power consumption. With DozyAP, a softAP can automatically sleep to save power when the network is idle and wake up on demand if the network becomes active again.

Putting a softAP into sleep imposes two challenges. First, without a careful design, it may cause packet loss. Existing Wi-Fi clients assume that APs are always available for receiving packets, so whenever a client receives an outgoing packet from applications, it will immediately send the packet to its AP. However, if the AP is in the sleep mode, this packet will be lost, even after the retries that occur at the low layers of the network stack. Second, putting an AP to sleep will introduce increased network latency and may impair user experience if the extra latency is user perceivable.

DozyAP addresses the first challenge with a sleep request-response protocol with which a softAP and its clients negotiate and agree on a valid sleep schedule. To avoid possible packet loss, a client will transmit packets only when the softAP is active and buffer outgoing packets otherwise. To address the second challenge, we design an adaptive sleep scheme and limit the maximum sleep duration. Consequently, DozyAP is able to reduce power consumption of Wi-Fi tethering with negligible impact on the network performance. DozyAP does not require any changes to the 802.11 protocol and is incrementally deployable via software updates to mobile devices.

We have implemented the DozyAP system on existing commercial smartphones and evaluated its performance using various applications and the traces from real users. Evaluation results show that DozyAP can put the Wi-Fi interface of a softAP to sleep for up to 88% of the total time in several different applications. Due to the restricted programmability of current Wi-Fi hardware on smartphones, forcing a softAP to sleep or wake up consumes considerable overhead. Thus, DozyAP only saves power by up to 33% while increasing network latency by less than 5.1%.

It is noticed that tethering can be also enabled by USB, bluetooth and Wi-Fi in ad-hoc mode. However, USB tethering has drawbacks that can only support one client. Also, connecting phones to devices such as tablets is not easy due to the constrained interface and complicated system configuration. Bluetooth suffers high energy consumption per bit transmission cost and low bandwidth [74], thus consuming more energy than Wi-Fi. Ad-hoc mode of Wi-Fi is less used than the infrastructure mode in practice. The OS on many mobile devices including Windows phones, Android phones and iPhones hides such a mode, preventing a device from connecting to an ad hoc network [87]. Due to the above reasons, those tethering methods are out of scope of this dissertation.

To the best of our knowledge, we are the first to study the power efficiency of a softAP in Wi-Fi tethering. The main contributions of this chapter are:

- We study the characteristics of existing Wi-Fi tethering and present our findings. We show that current Wi-Fi tethering is power hungry, wasting energy unnecessarily. We analyze the traffic patterns of various applications and identify many opportunities to optimize the power consumption of Wi-Fi tethering.

- We propose DozyAP to improve power efficiency of Wi-Fi tethering. We design a simple yet reliable sleep protocol to schedule a mobile softAP to sleep without requiring tight time synchronization between the softAP and its clients. We develop a two-stage adaptive sleep algorithm to allow a mobile softAP to automatically adapt to the traffic load for the best sleep schedule.

- We implement DozyAP system on commercially available off-the-shelf (COTS) smartphones and evaluate its performance through experiments with real applications and simulations based on real user traces. Evaluation results show that DozyAP is able to significantly reduce power consumption of Wi-Fi tethering and retain comparable user experience at the same time.

The rest of the chapter is organized as follows. In Section 4.2 we report our findings on existing Wi-Fi tethering, with focus on the power consumption and the traffic patterns of various applications. Based on the findings, in Section 4.3 we design DozyAP that can schedule a mobile softAP to sleep, and present the design details. We describe our implementation in Section 4.4 and evaluate it in Section 4.5. We discuss limitations of DozyAP and future work in Section 4.6, survey the related work in Section 4.7, and conclude in Section 4.8.

4.2 Understanding Wi-Fi Tethering

In this section we report our findings on the characteristics of Wi-Fi tethering through real measurements on existing commercial smartphones. We focus on two characteristics: the power consumption and the traffic pattern of various online applications used in Wi-Fi tethering. Furthermore, we provide some background on Wi-Fi power management to set up the context of our DozyAP design.

4.2.1 Power Consumption

We first measure the power consumption of existing commercial smartphones regarding the Wi-Fi tethering. We impose *no traffic* but simply turn on/off the Wi-Fi tethering, i.e., the Wi-Fi interface and the 3G interface were kept on but idle. We used a Nexus One phone (Android 2.3.6), a HTC HD7 Windows Phone (Windows Phone 7.5) and an iPhone 4 (iOS 4.3.5) for experiments. For the Nexus One and the HTC HD7, we measured the power consumption of the whole system using a Monsoon Power Monitor [12]. However, it is impossible to use the Monsoon Power

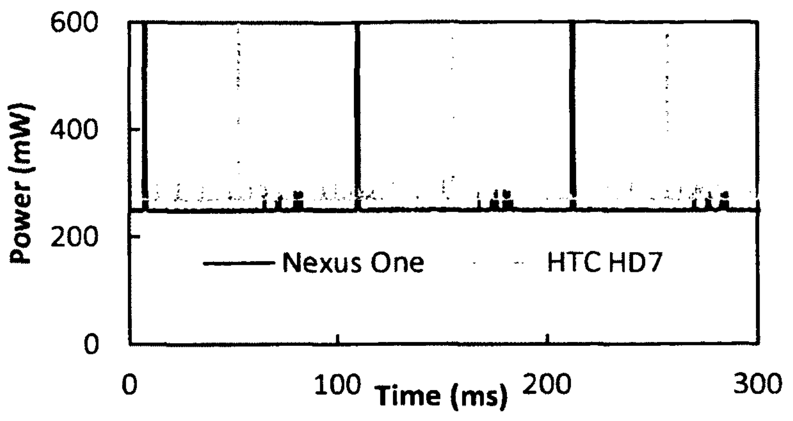


Figure 4.2: Measured power consumption of Wi-Fi tethering on Nexus One and HTC HD7 without background traffic

Monitor to measure the power consumption of the iPhone without damaging the phone. Instead, we used MyWi 5.0 [13], a very popular third-party Wi-Fi tethering tool on iOS that is able to tell the draining current of the battery, to measure the power consumption of the iPhone 4 when Wi-Fi tethering is enabled. In all the experiments, the display was turned off.

With Wi-Fi tethering disabled, the power consumption of all smartphones was pretty low, because the Wi-Fi and 3G interfaces were in sleep for most of the time. The average power consumption was only 20mW for the Nexus One and 30mW for the HTC HD7, respectively. For the iPhone 4, MyWi read a draining current of 6mA, equivalently a power consumption of 22mW.

With Wi-Fi tethering enabled, the power consumption of the smartphones increased significantly. Figure 4.2 shows the results of the Nexus One and the HTC HD7 smartphones. We can see that both smartphones operated in a high power state constantly even though there was no traffic at all. There are periodic spikes in the plots, caused by periodic Wi-Fi beacon transmissions. On average the power consumption was 270mW for the Nexus One and 302mW for the HTC HD7. For the iPhone 4, MyWi read a draining current of 90mA, equivalently a power consumption of 333mW. While software reading may not be as accurate as the Monsoon

Power Monitor, the result still clearly indicates that Wi-Fi tethering on the iPhone 4 has similar power consumption as on the Nexus One and the HTC HD7. Also, we tested phones with the latest OS versions such as Android v4.x. The power consumption was similar.

The above results demonstrate that existing Wi-Fi tethering schemes on all the three mobile platforms are power hungry. They consume *an order of magnitude more power than necessary* when there is no ongoing traffic, i.e., in idle network state. In next subsection we will show that such idle states occurs frequently in various typical Internet access scenarios.

Intuitively, the Wi-Fi interface should be put to sleep when the Wi-Fi network is idle. As the battery is a very scarce resource on smartphones, this calls for a power-efficient Wi-Fi tethering solution and motivates us to conduct the work in this chapter.

4.2.2 Traffic Pattern

Next, we study how frequently the Wi-Fi network is actually in an idle state and how long the idle state typically lasts. We enabled Wi-Fi tethering on a Nexus One smartphone with a China Unicom 3G connection. A Wi-Fi client is connected to such a mobile softAP. On the client side, we launched various applications to access the Internet and they are used normally. In the meantime, we used a Lenovo T61 laptop running Linux 2.6.32 as a Wi-Fi sniffer to capture all the packets exchanged between the client and the softAP. We studied two different clients: a Nexus One smartphone and a Wi-Fi version iPad 2. Seven applications were measured including news reading, online book reading, video streaming, search, Map, email and RSS.

Note that some websites detect the type of client devices and return different content for different device types. For example, when the Nexus One smartphone

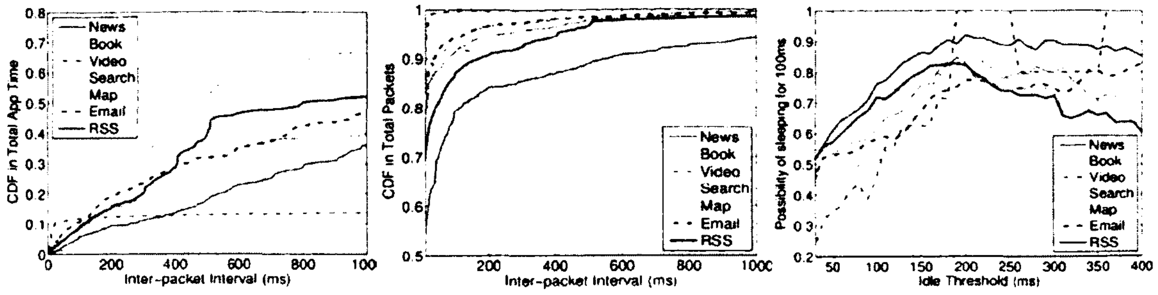


Figure 4.3: Traffic pattern of seven applications. From left to right: CDF of packet inter-arrival intervals in total application time; CDF of packet inter-arrival intervals in total packets; Probability of sleeping for 100ms after an idle threshold.

is used, Baidu News automatically redirects to its mobile version that returns less complex webpages than the normal version. Similarly, Youku streams low bitrate video clips to the Nexus One smartphone but high bitrate ones of the same videos to the iPad 2. As a result, the same application may behave differently on different devices. For each application, we study the traffic patterns by analyzing the packet inter-arrival time of all the captured packets.

Figure 4.3 shows the results of the Nexus One. Due to the space limitation, we omit the iPad 2 case which also has similar results. We first study the distribution of packet inter-arrival intervals in the total application time which indicates the period from the first packet to the last one. The left figure in Figure 4.3 shows the Cumulative Distribution Function (CDF) for all the applications, where the y-axis depicts the percentage of packets with inter-packet intervals less than or equal to a specific value in the x-axis to the total application time. To make the curves easy to read, we only show the data for the time intervals less than one second. We can see that the intervals under 200ms only take less than 30% of the total application time for all the applications on the Nexus One. For the iPad 2, the corresponding number is 35%. For some applications, these intervals consume as low as 20% or even less than 10% on the Nexus One or the iPad 2. If we consider the network "idle" during the packet inter-arrival intervals larger than 200ms, then we can say that the

Wi-Fi network was idle for 70%-90% of the total application time. This shows that these applications only spent a small portion of time for the Internet access and their network traffic is very sparse and bursty.

There are two main reasons for the above findings. First, all the applications consist of two phases: a content fetching phase and a content consuming phase. Once users download some content from a remote server (e.g., a Web server), they need to spend time to consume the content (e.g., reading the text). The content consuming time may vary from seconds to tens of seconds to even minutes. During such a time, the network is mostly idle. In the email case, replying to emails and composing new ones also result in significant network idle time. Secondly, the bandwidth of 3G is much lower than that of Wi-Fi. According to 3GTest [45], 3G typically offers 500Kbps-1Mbps downlink throughput for US carriers but the Wi-Fi offers much higher data rates (54Mbps for 802.11a/g and 300Mbps for 802.11n). Furthermore, 3G has much higher RTTs, ranging from 200ms to 500ms [45], than that of Wi-Fi. Consequently, the Wi-Fi interface of a softAP in Wi-Fi tethering often has to wait for data to be received from or transmitted over 3G. Such a waiting period will put the Wi-Fi interface in an "idle" state.

While the results are somehow as expected for those interactive applications, we are surprised to see that similar patterns were observed in the video streaming case. Even for the iPad 2 on which a high bitrate video clip was continuously played back, the packet inter-arrival intervals larger than 200ms took more than 60% of the total streaming time. After carefully checking the captured trace, we found that it used a large video buffer when streaming video clips. It aggressively downloaded video content until the video buffer was full. Then it stopped video downloading. The downloaded bits were constantly consumed and drained from the buffer. Once the buffer level became lower than a threshold, the aggressive downloading was resumed again.

The large percentage of the Wi-Fi idle time in these applications demonstrates that there are many opportunities to reduce the power consumption of Wi-Fi tethering. During the large network idle intervals, the Wi-Fi interface of a mobile softAP should sleep to save power. More specifically, there are two kinds of network idle intervals that we exploit in this chapter. The first one is the long network idle intervals resulting from the user content consuming behavior. The second one is the relatively shorter network idle intervals that occur during the content downloading. The latter case is mainly caused by the RTTs of 3G: after a client sends a request packet to a remote server, it has to wait for at least a RTT of 3G to get the first response packet from the server. For example, to access a Web server, we can typically see two such network idle intervals: one for the DNS name lookup for the server and the other for making a TCP connection to the server.

We further study how putting a softAP to sleep can affect the network performance. The middle figure in Figure 4.3 shows the CDF of packet inter-arrival interval in total packets, where the y-axis depicts the percentage of the packets whose inter-packet interval is less than or equal to a specific value in the x-axis to the total number of packets. We can see that the inter-packet intervals under 150ms cover more than 80% of all the packets for all the applications. For some applications the number is as high as 90% or even 95%. This means that if we use an idle threshold of 150ms to decide whether to put the softAP to sleep or not, most of the packets will not be affected. The right figure in Figure 4.3 further shows the probability that the softAP could successfully sleep for extra 100ms after waiting for different idle thresholds. We found that 150ms was a good threshold to optimize the energy saving and minimize the incurred network latency in terms of sleeping probability and the number of involved packets.

All the above findings demonstrate that a mobile softAP *could* and *should* sleep to save power in Wi-Fi tethering, which provides the foundation for our DozyAP design.

4.2.3 Background: Wi-Fi Power Saving

The IEEE 802.11 standard defines a Power Saving Mode (PSM) to save power for Wi-Fi clients [16]. In PSM, the Wi-Fi interface of a client always stays in a very low power state to save power and cannot receive or transmit any data. If an AP needs to send some packets to a client in PSM, the AP will first buffer the packet and set the Traffic Indication Map (TIM) in its beacons, which are broadcasted typically every 100ms. A PSM client periodically (i.e., every a certain number of beacon intervals) wakes up to listen to beacons. If the client detects a TIM for itself, it sends a individual PS-Poll frame to notify the AP of sending a buffered packet. Otherwise, it goes to sleep immediately. When the AP transmits a buffered packet to the client, a MORE flag in the header of the data frame is set if the AP has more packets for the client. This allows the client to decide when to stop sending PS-Poll frames.

On the Nexus One, the above static PSM scheme is called "PM_MAX". PM_MAX allows a client to sleep as long as the AP does not have any packet for it. However, this leads to long network latency and hence low network efficiency. Therefore, on the Nexus One, another power saving scheme called "PM_FAST" is used. In PM_FAST, a client stays in active unless its Wi-Fi interface is idle for a threshold of 200ms. Then it sends a Null-Data frame with power management flag set to 1 to tell its AP that it will sleep soon. If such a frame is acknowledged, the client is able to go to sleep since all packets destined for it will be buffered at AP. Otherwise, the client cannot go to sleep. Once the client detects a TIM for itself from beacons, it notifies the AP that it is active and ready to receive packets by a single Null-Data frame with power management flag set to 0. Many other Wi-Fi devices today also implement a similar scheme known as adaptive PSM [50]. PM_FAST is designed

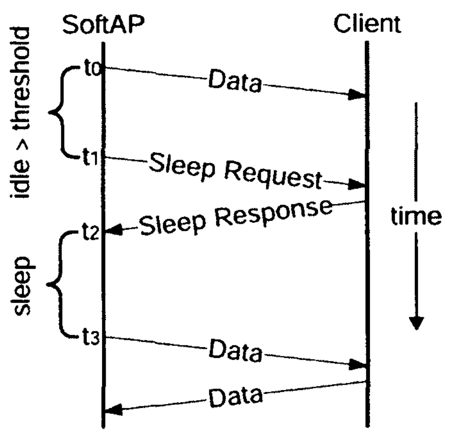


Figure 4.4: Interaction of a softAP and a client using the sleep request-response protocol

for fast system response and PM_MAX is more suitable for background services. By default Nexus One smartphones use PM_FAST if the screen is on and switch to PM_MAX if the screen is turned off.

4.3 DozyAP Design

Guided by the findings in Section 4.2, we design the DozyAP system that aims to reduce the power consumption of Wi-Fi tethering by putting the Wi-Fi interface of a mobile softAP into sleep mode whenever possible. Below we present the detail of DozyAP and the rationale of the design decisions. We start with a single client and describe the extension to support multiple clients later on.

4.3.1 Scheduling a SoftAP to Sleep

We design a simple *sleep request-response* protocol to enable a mobile softAP to safely sleep in Wi-Fi tethering according to its own best schedule. While the softAP can sleep at will, it can only do so when the client agrees, to avoid possible packet loss. Therefore, before entering sleep mode, a softAP sends a *sleep request* to its client. If the client sends back a *sleep response* to accept the sleep request, the softAP then enters the sleep mode. Otherwise, it will continue to stay in the active

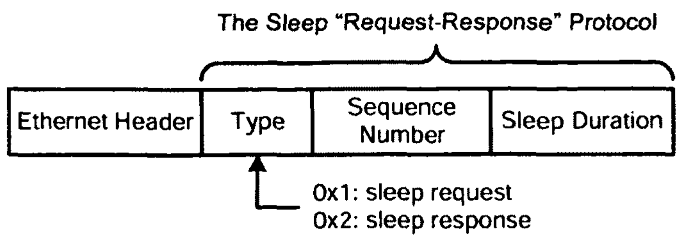


Figure 4.5: Packet format of the sleep protocol.

state. Figure 4.4 shows a typical interaction procedure between a softAP and a client. At time t_1, the softAP decides to sleep and enters sleep mode at time t_2 after receiving the client's agreement. When the sleep times out at time t3, the softAP wakes up and continues to communicate with the client.

Packet format. Both the sleep request and the sleep response are transmitted as a normal Wi-Fi unicast data packet. This design does not require any modification on existing Wi-Fi standard and is easy to implement. Figure 4.5 shows the packet format. The sleep protocol is implemented directly on top of the underlying link layer without TCP/IP headers in the middle to reduce the overhead. The sleep protocol packets have three fields. The "Type" field indicates the packet type: "0x1" means *sleep request* and "0x2" means *sleep response*. The "Sequence Number" field is a unique ID to identify a sleep request-response pair. It starts from zero and increases by one for every new sleep request. The "Time Duration" field specifies how long (in milliseconds) the softAP requests to sleep. All the sequence numbers and time durations are decided by the softAP. When the client accepts a sleep request, it simply copies the sequence number and time duration from the sleep request packet into its sleep response packet. Sleep response packets are used only for accepting a sleep request. If the client does not agree the softAP to sleep, it simply chooses not to send out the sleep response. There is only one case that the client will decline the sleep request of the softAP: it has more data packets to transmit. In that case, the client will send a data packet, instead of the sleep response packet, to the softAP. The softAP then learns that the client has declined the sleep request and thus stays active. This design reduces the overhead of the sleep protocol because a sleep response packet is transmitted only when it is necessary.

State machine. Figure 4.6 shows the state machine of a softAP and a client. A softAP has three states: *Normal*, *Pre-sleep* and *Sleep*. In the *Normal* state, the softAP is active and can transmit and receive packets normally. When the Wi-Fi interface of the softAP is idle for a time period larger than a pre-defined threshold, the softAP sends a sleep request packet to its client with a sequence number seq and a time duration dur. Then it enters the *Pre-sleep* state and waits for a sleep response. If it receives the right sleep response with the same sequence number seq and time duration dur, it will put its Wi-Fi interface into sleep mode and enter the *Sleep* state; if it receives any packet other than the expected sleep response, it will go back to the *Normal* state and invalidate the sleep request. In the *Sleep* state, the Wi-Fi interface of the softAP is turned to sleep to save power. Thus, the softAP cannot receive any packets over Wi-Fi. If it receives any data from its 3G interface, it will buffer them during the whole period of *Sleep* state. When the sleep timeout expires and some data are buffered, the softAP wakes up its Wi-Fi interface, switches to *Normal* state, and transmits the buffered data to the client. Otherwise, it moves back to the *Pre-sleep* state, sends out another sleep request with a new sequence number new_seq and a new time duration new_dur, and waits for the next sleep response.

The state machine of a client has only two states: *Normal* and *Block*. In the *Normal* state, the client communicates with the softAP as normal. It may use any Wi-Fi power saving schemes such as PM_MAX, PM_FAST or none. If the client receives a sleep request from the softAP and agrees, i.e., it does not have any packets to transmit, it tentatively sets its Wi-Fi power saving scheme to PM_MAX, sends back a sleep response to the softAP, and enters the *Block* state. Note that by switching to PM_MAX, the firmware automatically sends a Null-Data frame to tell

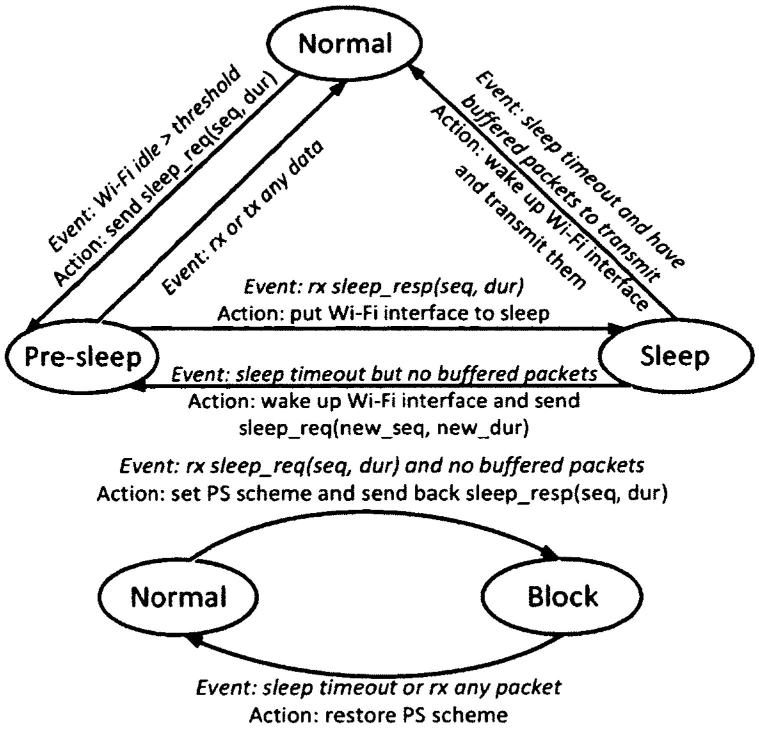

Normal
Event: Wi-Fi idle > threshold
Action: send sleep_req(seq, dur)
Event: rx or tx any data
Event: sleep timeout and have buffered packets to transmit
Action: wake up Wi-Fi interface and transmit them
Event: rx sleep_resp(seq, dur)
Action: put Wi-Fi interface to sleep
Pre-sleep
Sleep
Event: sleep timeout but no buffered packets
Action: wake up Wi-Fi interface and send sleep_req(new_seq, new_dur)
Event: rx sleep_req(seq, dur) and no buffered packets
Action: set PS scheme and send back sleep_resp(seq, dur)
Normal
Block
Event: sleep timeout or rx any packet
Action: restore PS scheme

Figure 4.6: State machine for a softAP (top) and a client (bottom)

the softAP that the client is going to sleep. Do this way, the client can go to sleep as quickly as possible (i.e., immediately after the sleep response). In contrast, if a client that uses PM_FAST does not change to PM_MAX before sending the sleep response, it will wait for 200ms idle period to send out a Null-Data frame. However, as the softAP has already entered the *Sleep* state once receiving the sleep response, it cannot receive the Null-Data frame afterwards. As a result, the Null-Data frame is not acknowledged and the client cannot go to sleep as supposed. Therefore, it is essential for clients to switch to PM_MAX to maximally save power. In the *Block* state, the Wi-Fi interface of the client is in power saving mode and the client knows that the softAP is sleeping. Thus, it blocks all the packet transmissions by buffering all the packets from applications. If the sleep schedule times out or the client receives a data packet from the softAP, it restores the previous power saving scheme (e.g., back to PM_FAST) and moves back to the *Normal* state. At this time, both the softAP and the client can communicate normally. Otherwise, the softAP will send a new sleep request to try to sleep again.

4.3.2 Synchronization

One advantage of the sleep request-response protocol is that it does not require tight time synchronization between the softAP and the client. If the softAP and the client can synchronize their time perfectly, they can coordinate their sleep scheduling to avoid packet loss without transmitting any extra sleep request and response packets. However, this is hard to achieve in practice. Although very fine-grained hardware timestamps (e.g., at microsecond granularity) exists at link layer, such timestamps are segregated inside firmware and are not available to the Wi-Fi driver and applications. It is possible to do time synchronization by explicitly exchanging packets with timing information between the softAP and the client. Such time synchronization must be done periodically due to clock drift, which increases power consumption. Due to these considerations, we intentionally avoided the time synchronization approach.

Interestingly, the proposed sleep protocol can achieve loose synchronization between a softAP and a client, with a desirable property: *the client will never conclude that the softAP is awake while it is sleeping*. Therefore, our approach will not lead to packet loss that would arise from wrong attempts of sending packets while the softAP is actually in sleep mode. In normal case, this is obvious because the softAP will sleep only after it receives a sleep response from the client. However, due to the uncertainty and complexity of wireless communication, the sleep protocol may not work as smoothly as expected. Below we analyze several possible abnormal cases and their consequences, as illustrated in Figure 4.7.

Packet loss. First, sleep request or response packets may be lost during their transmissions. For example, a sleep request may be lost. In this case, the softAP will stay in active. If later on the client or the softAP has data to transmit, they start

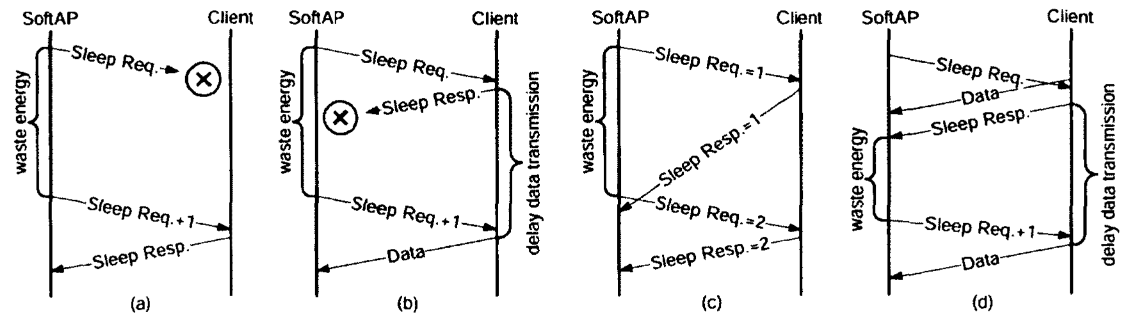


Figure 4.7: Abnormal cases. (a) a sleep request is lost. (b) a sleep response is lost. (c) a sleep response is delayed. (d) a data packet is delayed.

to communicate as normal. Or the softAP will send out a new sleep request after the network remains idle for a period longer than the pre-defined idle threshold, as shown in Figure 4.7(a). The worst effect of losing a sleep request is that the softAP would waste some energy for staying in unnecessary active state between two successive sleep requests. Similarly, if a sleep response is lost, the softAP also has to stay in active until the next sleep request. However, in this case, as the client has concluded that the softAP is in sleep, it will stay in the Block state and start to buffer packets. Thus, it may further incur extra delay up to the idle threshold to the client, as shown in Figure 4.7(b).

Packet out-of-order. Second, packet transmission may be delayed due to the hardware queuing and wireless contention. As a result, there is a slight chance that packets may not arrive at their destinations in the expected order. For example, Figure 4.7(c) shows the case that a sleep response is delayed by the client□s hardware. The softAP receives sleep response 1 after sleep request 2 is sent out. In this case, the softAP just ignores sleep response 1 but it has to stay in active between the two sleep requests. Figure 4.7(d) shows a more complex case. The client has already passed a packet to the firmware and the packet is waiting for transmission in the hardware queue. At this moment the client receives a sleep request from the softAP. As the client does not have more data to transmit, it replies a sleep response. However, once the softAP receives the data packet, it resets its idle timer, stays in active and ignores the sleep response. Consequently, the softAP and the client are out of sync: the softAP stays in active, wasting energy, but the client assumes the softAP is in sleep and delays its packets transmission.

Based on the above analysis, we can see that those abnormal cases would at most cause some overhead in energy and transmission latency, but they would not break the desired synchronization property of our sleep protocol. A client will never try to send packets when the softAP is actually in sleep. The softAP and the client may run out of sync temporally, but will always resume sync after the subsequent sync response. This demonstrates the robustness of our sleep protocol. In addition, in the Wi-Fi tethering, the bottleneck is usually the 3G connection as its bandwidth is much lower than that of Wi-Fi. The necessity airtime of Wi-Fi is usually light, and thus, the above abnormal cases can rarely happen.

It is noticed that the sleep request-response protocol operates above the MAC layer, thus all the MAC layer frames such as beacons and Null-Data cannot be seen by the protocol. That is why DozyAP has to explicitly exchange sleep requests and responses to negotiate a sleeping schedule. If the protocol is applied to the MAC layer, the negotiation can be performed through existing MAC layer frames. For example, if a client does not have data to transmit, it sends a null-data frames with power management flag set to 1. Once observing that all clients have been in sleep mode, the AP automatically turns off. Owing to the tight time synchronization existing at MAC layer, both AP and clients could wake up almost at the same time when the pre-determined sleep timeout or next beacon is due. Then AP could send buffered packets and clients could upload buffered packets.

4.3.3 Adaptive Sleeping

Our sleep protocol allows a softAP to sleep. The next natural question is: how long should it sleep? The simplest solution is certainly to sleep for a fixed interval. However, it is difficult to determine such an interval, because the RTT of 3G connection varies and the packet arrival time is irregular. In our design, we come up with an adaptive sleep scheduling algorithm to adapt to the traffic pattern and also the 3G network property. Our adaptive sleep algorithm consists of two stages, namely a *short sleep* stage and a *long sleep* stage, that are designed to exploit the two distinctive phases (i.e., the content downloading phase and the content consumption phase) of interactive applications, respectively.

Sleep algorithm. Figure 4.8 shows how the two-stage adaptive sleep algorithm works. The basic idea is to probe the optimal sleep interval such that the softAP can wake up shortly before a packet arrives. Starting with an initial conservative sleep interval, the sleep interval is gradually increased, at a conservative pace, until a packet has arrived during the last sleeping. Then the initial sleep interval is updated dynamically. For sake of easier expression, all the successive sleep slots are collectively called a sleep cycle.

More concretely, when the Wi-Fi interface remains idle for a time period of *thresh*, the softAP will enter the short sleep stage. It first sleeps for a time period of *init* which equals to *min* initially. When the softAP wakes up, it either goes back to the ACTIVE mode if there are pending outgoing or incoming packets, or continues to sleep for a fixed interval of *step*. Depending on the real packet arrival pattern, the length of the sleep cycle may become longer and longer between two subsequent wake-ups. The sleep period can be expressed as $init + N * step$ where N is the number of continuous sleep slots after the first sleep slot of *init*.

As waking the Wi-Fi hardware up introduces certain energy overhead [59], it is desirable to reduce the number of unnecessary wake-ups. This calls for a good *init* value that can let the softAP sleep as long as possible while still being able to wake up in time, i.e., to avoid or shorten the probing process. We determine the *init* value by exploiting the sleep history. We use parameters *cur* and *pre* to track the gross

```
// Parameters:
thresh, min, max, init, step, cur, pre, thresh_l, long;
ACTIVE:
measure the Wi-Fi network idle time;
SHORT_SLEEP:
if (Wi-Fi network idle time > thresh)
  first = true;
  sleep for a time period of init;
while (1)
  cur = first? init : (cur + step); first = false;
  if receive or transmit a packet
    if (cur <= init)
      init = max(init - step, min);
    pre = cur; goto ACTIVE;
  if ((cur > init + step) && (pre > init + step))
    init = min(init + step, max);
  if (cur >= thresh_l)
    pre = cur; goto LONG_SLEEP;
  sleep for a time period of step;
LONG_SLEEP:
while (1)
  sleep for a time period of long;
  if receive or transmit a packet;
    goto ACTIVE;
```

Figure 4.8: The two-stage adaptive sleep algorithm.

length of all successful sleep slots in the current sleep cycle and that in the previous sleep cycle, respectively. That is, we have *cur* equals to $init + (N-1) * step$ because a sleep cycle is always ended up by a false sleep slot during which a packet has arrived and been buffered. Parameter *pre* is simply a running record of the previous *cur*. Based on the values of *cur* and *pre*, we adjust the value of *init* with a simple algorithm INIT_UPDATE as follows: if both *cur* and *pre* are greater than current *init* plus *step*, we increase $init$ by $step$ for the next sleep cycle. If cur is less than or equal to current $init$ minus $step$, we decrease $init$ by $step$. To avoid excessive latency that may be caused by an overly greedy $init$ value, we cap it to the value of max. In SHORT_SLEEP stage, if the softAP has continuously been in sleep for a time period of $thresh\_l$, it goes to LONG_SLEEP stage. In LONG_SLEEP stage, the softAP simply sleeps for a time period of $long$ periodically until it quits from sleep

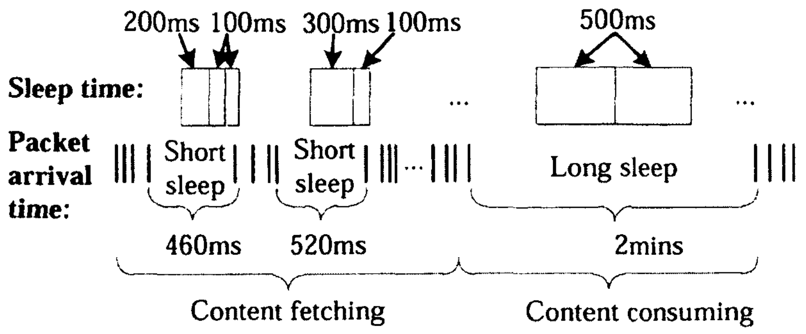


Figure 4.9: Short sleep and long sleep example.

to communicate with the client.

Example. Figure 4.9 illustrates the algorithm with a concrete example. Some details such as the time for waking up the Wi-Fi interface between continuous sleep slots are omitted for sake of easier reading. Assume current value of *init* is 200ms and the value of *step* is 100ms, in the first short sleep circle (the 460ms one), after *thresh* (150ms) idle time for triggering sleep request-response protocol, the softAP will first sleep for 200ms, followed by two 100ms sleep slots. Suppose the value of *init* is then qualified to increase to 300ms. In the second short sleep circle, after 150ms idle time, the softAP will first sleep for 300ms followed by one more 100ms sleep slot. In the content consuming period, after sleeping for a time period of *thresh_l*, the softAP enters the long sleep stage and periodically sleeps for a time period of *long* (500ms).

The above algorithm is specially designed for the traffic patterns of typical applications in Wi-Fi tethering as shown in Section 4.2.2. The short sleep stage is designed for the softAP to sleep between the time when the client sends out a request to a remote server and the time when the first response packet from the remote server is received. That duration is roughly a RTT of the 3G connection (typically hundreds of milliseconds [45]). The purpose of *init* parameter is exactly to estimate the 3G□s RTT in an elegant way, based on the length of last two sleep cycles. Note that our algorithm is conservative in the sense that it tries to reduce the energy consumption under minimal impairment to user experience, i.e., extra latency incurred. We decrease the value of $init$ quickly, by considering only the length of the current sleep cycle, but increase the value of $init$ slowly by considering the length of both the current and the previous sleep cycles. In addition, we use the parameter $thresh$ to prevent the softAP from entering the short sleep stage during burst data transmission period (e.g., multiple response packets from a remote server for the same client request such as fetching a picture. In Section 4.4 we describe the parameter values used in our implementation.

In summary, our sleep algorithm automatically adapts to the traffic pattern of applications and achieves a good balance between power saving and network performance.

4.3.4 Supporting Multiple Clients

DozyAP can support multiple clients by repeatedly applying the sleep request-response protocol to each client. A client goes to sleep once it agrees to the AP's sleep request. A softAP can sleep only if it receives the sleep responses from all the clients. If some clients replied to a sleep response but other clients did not, the softAP has to stay awake in this case. This design makes sense because some clients may have data to send and the softAP must serve those clients. It is expected that the softAP sleeps less and consumes more power in the multi-client case. However, extending DozyAP to support multiple clients will not break the synchronization property of the sleep protocol: *no client will send a packet when the softAP is in sleep*. Note that we considered the possibility of broadcasting the sleep requests as it can obviously reduce the overhead of the sleep protocol. However, we do not take this approach for two reasons. First, broadcast packets are less reliable because they are transmitted without link layer retransmissions. Second, the clients in PSM likely cannot receive broadcast packets, whereas the unicast

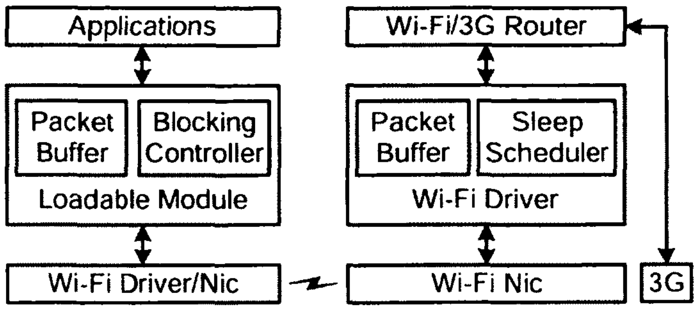


Figure 4.10: Implementation architecture of the client part (left) and the softAP part (right).

packets will be buffered in AP's hardware transmission queue until clients wake up from PSM. Therefore, the improvement of using broadcast is expected to be very small in multi-client case, and for single client case, it is worse than using unicast.

Another minor issue with the multi-client case is the beacon. In our design a softAP does not send out beacons in the sleep mode. Thus, a new client cannot join the Wi-Fi network when the softAP is in sleep. However, the softAP sends out periodic beacons when it is active. Even in long sleep stage, it still wakes up periodically and can send out beacons. Consequently, a new client is still able to find the softAP but may experience slightly longer latency. As this only happens when a new client joins the network, we think it is acceptable.

4.4 Implementation

We have implemented the DozyAP system on a Nexus One smartphone running Android 2.3.6, with a Wi-Fi chipset of Broadcom BCM4329 802.11 a/b/g/n [27].

The overall architecture consists of two parts: the softAP part and the client part, as shown in Figure 4.10. The softAP part is directly modified from the open source Wi-Fi driver in which we embedded the sleep request-response protocol and the two-stage adaptive sleep algorithm. When the softAP is in sleep state, all the packets received from 3G interface are buffered. The client part is implemented as a loadable module where a packet buffer is implemented, together with a blocking controller to decide if and when application packets must be buffered. In our prototype we use a special Ethernet type of `0xffffff` (a reserved value that should not be used in products) for the packets of the sleep request-response protocol. In real deployment, other approaches can be used to implement the sleep protocol, e.g., using dedicated IP packets rather than the special Ethernet type. We use *netfilter* to intercept all the outgoing packets and to detect the packets of sleep requests and response. Implementing the client part as a loadable module does not require any modifications to the source code of the client OS. This makes it easy to deploy DozyAP on different types of client devices.

Putting a mobile softAP into sleep. One practical difficulty we met is how to put a mobile softAP to sleep. On smartphones (Nexus One, and other types of smartphones), most Wi-Fi MAC layer functionalities are implemented in the firmware running on the Wi-Fi chipset, not in the CPU-hosted Wi-Fi driver. When Wi-Fi tethering is enabled, the firmware keeps the Wi-Fi always in a high power state. There is no interface available to change the power states. After trying many methods, all that we can do in the driver is to turn on/off the Wi-Fi interface when the softAP decides to wake up/sleep. By modifying the source of the driver, we hide the fact that the Wi-Fi interface is turned off. Thus, applications and the OS can work as normal as if the Wi-Fi interface is always on.

Energy overhead of turning on/off the Wi-Fi. It costs extra energy to switching on/off the Wi-Fi interface. We measured such energy overhead on a Nexus One smartphone and Figure 4.11 shows the measurement results. Initially, the Wi-Fi interface was off. Then we turned on the Wi-Fi interface for 100ms and turned it off again. We can observe two artifacts: First, when the Wi-Fi interface is merely turned on without transmitting any packet, the system stays in an average power state of 400mW which is higher than the normal power consumption of Wi-Fi tethering in idle case as shown in Figure 4.2. We think this part of overhead is caused

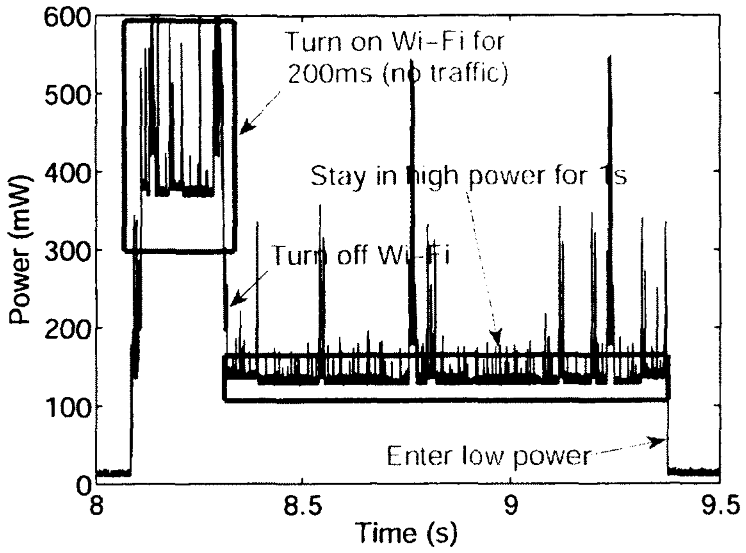


Figure 4.11: Power consumption of switching on/off the Wi-Fi interface.

by the CPU and I/O operations for waking up the Wi-Fi interface. Second, when the off command is issued, the power consumption reduced immediately, but remains at a power level as high as 150mW for about one second before entering a very low power state of 10mW. This finding is similar to what the authors reported in [59]. They pointed out that when the Wi-Fi interface goes to sleep, it first enters a "light sleep" state and then enters a "deep sleep" state after some time. We cannot control this behavior but it significantly affects how much power we can save. We want to point out that this is a platform-specific limitation caused by the restricted programmability over the Wi-Fi hardware. Our design itself does not impose any limitation. If the smartphone can incur less wakeup overhead or enter the "deep sleep" state more quickly, our approach can save much more power.

Parameter values. We determine the parameter values based on the real-world traces described in Section 4.2. All the parameters are set in a conservative way to handle the variations of network conditions. We set $thresh$ to 150ms for triggering a softAP to enter sleep mode. With this parameter, the chance of sleeping is maximized and the number of involved packets is limited to avoid introducing more network latency. Due to the overhead of wakeup, the sleep duration less than 100ms may not yield much power saving. Thus, we set min to 100ms. Considering the RTT of 3G and to limit the maximum extra latency, we set max to 500ms. The value of $init$ varies between 100ms and 500ms. We set $thresh\_l$ to 3s for switching to the long sleep stage where the softAP periodically sleeps for 500ms (i.e., $long$ equals to 500ms). It should be noticed that different Wi-Fi chipsets may have the different overhead of waking up, and thereby affecting the choice of the parameter values above. The guideline is as follows: If the overhead is larger, then the parameters should be adjusted to avoid sleeping too frequently. Otherwise, it is better to set to gain more power saving.

4.5 Evaluation

We evaluate the performance of DozyAP by answering the following questions. 1) How much power can DozyAP save for a mobile softAP in various applications? 2) What is the impact of DozyAP on client side power consumption? 3) How much extra latency does DozyAP introduce? 4) How much power can be saved in multi-client case? 5) What is the performance degradation if clients are not changed?

4.5.1 Experiment Setup

Hardware devices. We used a Nexus One smartphone as a softAP with a China Unicom 3G connection (WCDMA), and another Nexus One smartphone as a client to run applications. Both smartphones run Android 2.3.6. We used a Monsoon Power Monitor [12] to measure the power consumption. We repeated every experiment for at least five times to compute average results.

Applications and methodology. We used five of the applications described in Section 4.2, including news reading, book reading, video streaming, search, and

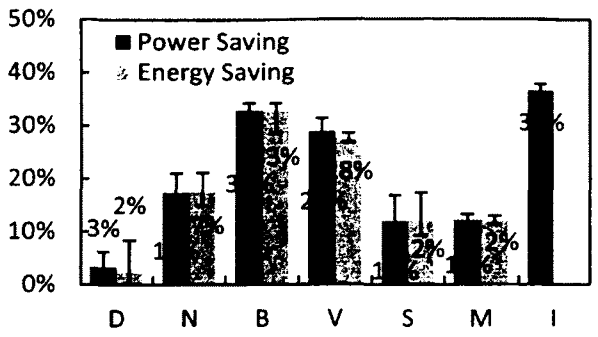


Figure 4.12: Power saving and energy saving of the softAP in idle (I), busy download (D), and the five applications of news reading (N), book reading (B), video streaming (V), search (S), and map (M)

map. To make the experiments repeatable, we analyze the captured trace of the applications to find out all the HTTP requests contained in the traces except video streaming. Then we wrote a test program in Java to send out those HTTP requests with the exact same order and timing as the traces. The program uses the WebView class in the WebKit package [4]. Thus, we could easily repeat every experiment. For the video streaming, we manually played the same video clip.

Traces. To evaluate DozyAP with more diverse and realistic traffic patterns, we asked the authors of MoodSense [53] for the traces collected from real users. In MoodSense, the authors conducted a two-month field study with 25 iPhone users and collected their network traffic everyday using tcpdump [78]. We selected the traces of the top eight most active 3G users. For each of them, we further selected the trace of the day when the user generated the largest 3G traffic volume. We used the eight-day traces to evaluate the performance of DozyAP.

4.5.2 Power Consumption

Average power. We first measured the power consumption of a mobile softAP with DozyAP and without DozyAP. Besides the five applications, we also measured two extreme cases: *idle* and *busy download.* In the idle case, we measured the power consumption of the softAP with one client associated but without any network traffic.

In the busy download case, we measured the power consumption of downloading a 1MB file from a Web server. The dark bars in Figure 4.12 show the average power saving of DozyAP. Without explicit mention, the error bars depict the minimum and maximum values in all the experiments. We see that DozyAP can reduce the average power by 12.2% to 32.8% for the five applications. In the idle case, it can save power by 36.5%. Even for the busy download case, the average power can be reduced by 3.3%. It is worth noting that the power saving percentage is calculated in *the total power consumption of the whole system*, including the power consumed by CPU and 3G. 3G consumes significant power when transmitting and receiving data. If we only consider the power consumption of Wi-Fi, the power saving percentage will be even higher in busy download and the five applications.

Total energy. As DozyAP buffers packets and delays their transmission, it may lead to longer application time comparing with the case without DozyAP. Thus, we also measured the total energy for the busy download case and the five applications. Total energy does not make sense for the idle case. The light bars in Figure 4.12 show the results. We see that DozyAP does not increase the total energy. Instead, it can save the total energy by 12.2% to 32.9% for the five applications, which is almost the same as the result of average power. As we show in Section 4.5.3, DozyAP indeed introduces very little network latency which has negligible impact on the total energy. Even in the busy download case, DozyAP can save the total energy by 2.3%.

Wi-Fi interface sleep time. As we point out in Section 4.4, with the current commercially available smartphones, forcing the softAP to go to sleep or wakeup can be only achieved by turning off/on the Wi-Fi interface. That results in significant overhead (see Figure 4.11). If we have more control on the power states of the Wi-Fi hardware (e.g., if we can directly modify the firmware or if we have a MadWifi [11] style driver which implements most MAC layer functions in driver rather than in

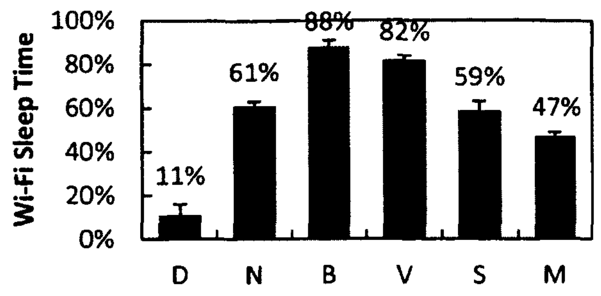


Figure 4.13: Wi-Fi interface sleep time of the softAP in idle, busy download and the five applications

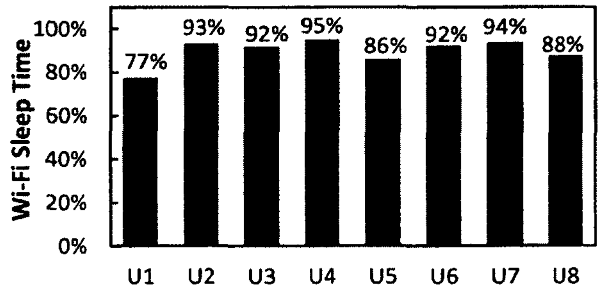


Figure 4.14: Wi-Fi interface sleep time calculated based on the traces of eight real users

firmware), DozyAP should be able to save significantly more power. We measured how much time DozyAP can put the Wi-Fi interface of a mobile softAP to sleep. Figure 4.13 shows the results. We see that the Wi-Fi interface of a softAP can stay in sleep mode for 47%-88% of the total time in the five applications. Even in the busy download case, we can turn the Wi-Fi interface to sleep for 11% of the total time. These results demonstrate the potential of DozyAP to significantly reduce the power consumption of Wi-Fi tethering. Given proper control over the Wi-Fi hardware, more energy is expected to be saved from sleeping.

We also evaluated the Wi-Fi interface sleep time with the real traces of the eight users in MoodSense [53]. To do it, we wrote a program to analyze the packet inter-arrival time of the traces and calculate the Wi-Fi interface sleep time as if these traces have happened in Wi-Fi tethering. To make the calculation reasonable, we ignored all the inter-packet arrival intervals larger than 5 minutes. That is, for any

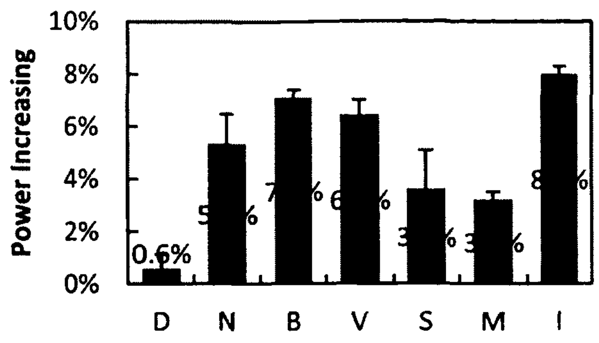


Figure 4.15: Power increasing of the Nexus One client in idle, busy download and the five applications

intervals larger than 5 minutes, we treated it as if the user stopped using the phone and turned Wi-Fi tethering off. This treatment is conservative because a user may spend more than 5 minutes to read a long news article or Wi-Fi tethering might not be turned off even the user stopped using the phone for 5 minutes. Figure 4.14 shows the calculated results. We see that DozyAP is able to allow the Wi-Fi interface of a softAP stay in sleep mode for 77%-95% of the time for the mixed multi-application user traffic. The numbers in Figure 4.14 are higher than the ones in Figure 4.13. The reason is that the experiments in Figure 4.13 focused on single application usage only. In practice users may use multiple applications one by one. Switching from one application to another leads to more network idle time.

Power consumption of a client. We also measured the power consumption of the Nexus One client in the idle case, busy download and the five applications. Figure 4.15 shows the results. We see that DozyAP can increase the power consumption of the client by less than 7.1% for these five applications. The reason is that the client needs to wake up to receive the sleep requests from the softAP and send back the sleep responses when the network is idle. Thus, the idle case introduces the highest overhead but it is still only 8%. Compared to the large power saving of the softAP, this small overhead is acceptable.

Multiple applications on single client. In some cases, multiple applications

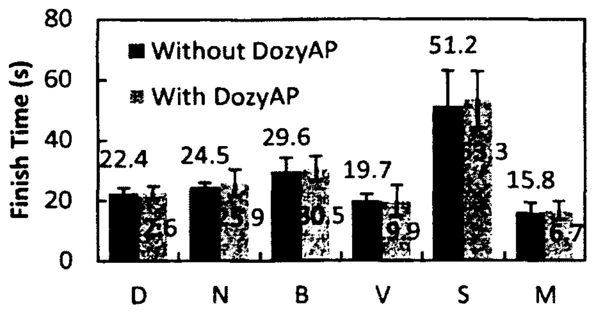


Figure 4.16: Finish time of busy download and the five applications

may run on a single client simultaneously. We evaluated DozyAP in a typical scenario where a user is reading news in the foreground meanwhile listening online music in the background. To do it, we first started *Douban FM* (which is a popular app in China like *Last.fm*). Once the music began to load, we started the news reading program (the same as before) immediately. The average power saving and energy saving over ten experiments is 14.5% and 14.2% respectively.

4.5.3 Latency

DozyAP incurs extra network latency because it delays packet transmissions when a softAP is in sleep mode. If the extra latency is user perceivable, it may impair user experience. As all the five applications are about fetching remote Web content, users care about the page loading time which is the period from the time when a user sends out a webpage request to the time when the webpage is fetched and rendered by the browser. The page loading time metric is widely used to evaluate the performance of browsers and Web servers. We evaluated the finish time of loading content which is the sum of the page load time of all the webpage requests in an application. The WebView object used in our test program could tell when a webpage is loaded. In the experiments, we sent out all the webpage requests of an application one by one without any time interval and calculated the total finish time.

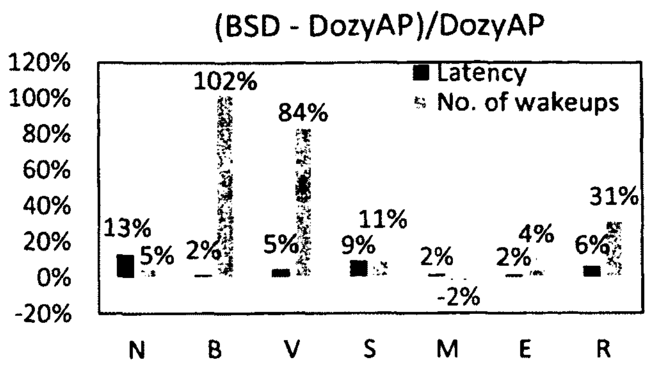


Figure 4.17: BSD vs. DozyAP in news reading (N), book reading (B), video streaming (V), search (S), map (M), email (E), and RSS (R)

Figure 4.16 shows the average result and the variance in busy download and the five applications. We see that DozyAP introduces very small extra network latency, ranging from 0.9% to 5.1%. Such small extra latency is hardly perceivable by users because of two reasons. First, as the 3G network has limited throughput and large RTT, it takes several hundred milliseconds to even seconds to load a webpage. Second, the time variance of the page loading time is pretty large, up to several seconds. That is, even without DozyAP, users already experience long page loading time with large variance. Therefore, the small latency increase of less than 5.1% is very hard to detect.

In addition, we compared our adaptive sleeping algorithm to BSD algorithm [50]. The reason why we choose BSD is that it is the-state-of-the-art algorithm that can adapt to the sleep duration without the MAC layer or lower layer support. To be fair, we investigated BSD and our algorithm based on on the same traffic traces collected from the user studies (as mentioned in Section 4.2). The results are shown in Fig. 4.17. As we see, BSD algorithm may introduce extra network latency and more wakeups in most cases. This extra latency may delay the application finish time. The increased number of wakeups may cost more energy due to the wakeup overhead.

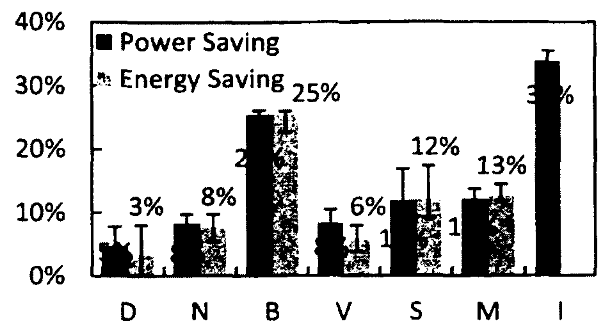


Figure 4.18: Power saving and energy saving of softAP with two clients

4.5.4 Multiple Clients

We first evaluated the performance of DozyAP with two clients associated: a smartphone and a tablet. Each client ran the same programs simultaneously. Figure 4.18 shows the average energy saving and power saving in busy download, the five applications and the idle case. As expected, the most power and energy savings are lower than the ones in single client scenario. However, the saving in download case does not drop as much as other applications. That is because no matter one or multiple clients were downloading, the cellular bandwidth was similarly saturated so that the chance for softAP to sleep is equivalent. Another finding is that the power saving for video streaming has a significant drop from about 28% to less than 10%. The reason is that two clients were competing in streaming video so that both of them needed more time to finish. Thus, the softAP had less opportunity to sleep.

We also conducted user studies to evaluate DozyAP with more clients. In the experiments, four clients with two phones, a tablet and a laptop were tethered to a DozyAP-enabled smartphone. They were asked to access Internet freely, such as reading news, checking and replying emails, listening Internet radios, and searching interesting places on Google Map. Since clients behave differently in each experiment, it is difficult for us to obtain the ground truth about the power con-

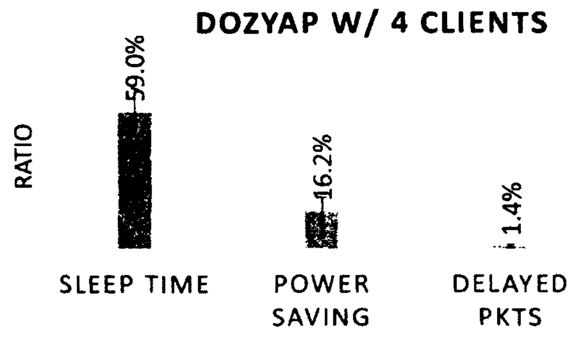


Figure 4.19: Performance of DozyAP w/ four clients. The ratios of sleep time and delayed packets are measured, whereas the ratio of power saving is approximated based on the ratio of sleep time.

sumption without DozyAP. Thus, we only measured the sleep time to approximate the power saving and count the buffered packets to show the incurred network latency. The results are shown in Fig. 4.19. We see that DozyAP allows the Wi-Fi to sleep for 59% of total application time and only causes about 1.4% delayed packets. From the experiments in Fig. 4.12 and Fig. 4.13, we observed that the *sleep-to-power-saving* translation ratio was around 18% to 36%. Therefore, the approximated power saving for our four-client tests is about 16.2% on average.

It is worth noting that the energy gain of DozyAP mainly depends on the traffic rather than the number of clients. Even if a single client is associated, it may generate continuously bursty traffic such as downloading. The power saving in that case is still less than the case of multiple clients but with sparse traffic.

4.5.5 Comparison with client-independent solution

To show the necessity of changing clients, we compared DozyAP to a straightforward client-independent solution, where the AP periodically wakes up after a sleep. If not receiving any data, the AP waits for a fixed time period and then goes back to sleep again. In this approach, the softAP wakes up and sleeps without notifying clients, so the packets transmitted by the clients may be lost. We use α to denote the ratio of the wakeup duration to the sleep duration and investigated the performance degradation with varying α.

We selected four applications including browsing websites (both 3G and regular websites), searching, downloading, and video streaming. The performance metrics are as follows:

- *Timeout ratio.* It depicts the percentage of the connection timeout reported by the client's web browser. The connection timeout is caused by the loss of HTTP requests when the client sent those requests but the softAP in sleep mode did not receive them.
- *Latency increasing ratio.* With respect to page (or download) finish time, latency increasing ratio presents the increased latency normalized by the benchmark finish time in DozyAP. A large ratio means that more delays are introduced by the client-independent solution mentioned above.
- *Traffic increasing ratio.* Due to the retransmission of lost packets, the client may send or receive more packets. Traffic increasing ratio is used to describe the percentage of the number of increased packets to the number of total packets.
- *Power saving ratio and energy saving ratio.* The meaning of these metrics are the same as mentioned before. Note that these metrics are measured on the AP side, whereas the above three metrics are measured on the client side.

Table 4.1 shows the averaged results over ten tests when $\alpha = 0.4$. Other values of α yielded worse results, so we do not present them here. As we see, without modifying the clients the timeout ratio is between 20%-40%, thereby impairing the user experience of Internet access. Both latency and traffic are heavily increased, where video streaming has the lowest increasing ratio and searching has the high-

	Timeout	Latency	Traffic	Power	Energy
Web(3G)	20%	46%	15%	20%	-11%
Web	40%	38%	7%	20%	-7%
Search	30%	74%	50%	29%	-26%
Download	40%	17%	4%	17%	-3%
Video	20%	12%	1%	16%	-3%

Table 4.1: Performance of client-independent solution

est increasing ratio. The power saving by the client-independent solution is competitive to softAP, because the Wi-Fi interface is also turned off periodically. However, due to longer application finish time, the client-independent solution actually spends more energy than existing Wi-Fi tethering schemes. Therefore, it demands modifying the clients to achieve better performance.

4.6 Discussion and Future Work

DozyAP requires patching the OS of smartphones working in Wi-Fi tethering and installing a loadable module on a client, which may be a hurdle for device vendors to overcome in practice. Despite this, we believe this problem is not very difficult to tackle, for instance, through Over The Air (OTA) upgrade. It also may be difficult to upgrade the software of dumb Wi-Fi client devices, e.g., music players and e-readers. However, to access the Internet, most people use "smart" devices including smartphones, tablets and laptops. All these devices are programmable and upgradable. Although our current implementation is based on Linux-style OS kernel including iOS, for Windows based devices a similar approach can be used via loadable Network Driver Interface Specification (NDIS) [14] driver.

DozyAP takes advantage of the speed discrepancy between cellular and Wi-Fi. One may argue that such an advantage will not exist when 4G is deployed. However, the speed of Wi-Fi increases fast too. With 11n and 11ac, there is still a big gap between cellular and Wi-Fi. In addition, our solution benefits not only from such a speed discrepancy, but also from the long content consuming time of users.

Our implementation uses fixed parameter values derived from the measurement results, which can be improved. For example, one may use a dynamic approach to tune the parameters to better adapt to the network conditions. Even though we use fixed values, we take a conservative way, e.g., the sleep time starts from a small value of 100ms. As shown in Figure 8, the tuning procedure of parameter $init$ is also conservative.

More power can be saved through transmission power adaptation. The built-in Wi-Fi tethering on existing smartphones always uses the highest transmission power. It wastes energy because a softAP is often close to its clients in Wi-Fi tethering. We plan to design a scheme to automatically adjust the transmission power based on the network conditions (e.g., RSSI and packet loss). We also plan to further take advantage of the bandwidth discrepancy between 3G and Wi-Fi to create more opportunities for a softAP to sleep. The basic idea is shaping the traffic between 3G and Wi-Fi. For downlink traffic, the softAP can buffer the packets received from 3G and send them to the client over Wi-Fi in batch. For uplink traffic, if the 3G connection is congested, the softAP can ask the client to stop sending more data. Thus, the Wi-Fi interface of both the softAP and the client can sleep longer.

DozyAP could be implemented in the MAC layer for further improvement if the Wi-Fi firmware is open on smartphones. The current implementation of DozyAP incurs the performance penalty from three aspects. First, explicit transmission of sleep request and response packets consume extra power for both AP and clients. Second, the overhead of switching on and off Wi-Fi interface is considerable. Third, additional power consumption is imposed by the CPU computation, since DozyAP has to involve the CPU to generate packets and run algorithms that are supposed to run on Wi-Fi chipsets.

4.7 Related Work

Wi-Fi power saving. There has been a lot of research effort devoted to power saving in Wi-Fi [19,23,33,44,50,55,59,68,70,72], focusing on improving the existing PSM in general or targeting at specific applications or usage scenarios. To name some recent work, Catnap [33] exploits the bandwidth discrepancy between Wi-Fi and broadband to save energy for mobile devices. NAPman [72] employs an energy-aware scheduling algorithm to reduce energy consumption by eliminating unnecessary retransmissions. SleepWell [59] coordinates the activity circles of multiple APs to allow client devices to sleep longer. All these solutions are for Wi-Fi clients only. DozyAP is complementary, focusing on the power efficiency of APs. Putting an AP to sleep is more challenging than putting a client to sleep because client devices expect that their AP is always on. To avoid packet loss, a softAP in DozyAP must coordinate its sleep schedule with its clients, which is different from existing work.

There is little work on power saving of APs. In [90,96], the authors propose to extend the IEEE 802.11 standard to support power saving access points for multi-hop solar/battery powered applications. Without building any real systems, they focus on protocol analysis and simulation, assuming Network Allocation Vector (NAV) can be used. Our work focuses on system design and implementation. We build real systems on commercial smartphones and do evaluation with real experiments. In addition, the NAV-based approach cannot work on existing smartphones because NAV is only visible in firmware. Cool-Tether [74] considers an alternative way to address the mobile hotspot problem that involves reversing the role of the phone and the client. However, it significantly increases the power consumption of the client and does not support multiple clients. In [29], the authors design algorithms to save power for APs with Wi-Fi Direct. Since Wi-Fi Direct is a separate mode on devices, it cannot be used for tethering till now. Furthermore, we measured the power consumption of current Wi-Fi Direct enabled devices, it is slightly higher than tethering.

Traffic-driven design. Adapting to traffic load for better sleeping is not a new idea [19,70]. Traffic patterns in different applications and scenarios have also been studied in some papers and the similar observations are identified (e.g., the large portion of network idle time) [50,55]. DozyAP builds on top of the basic techniques and applies them to Wi-Fi tethering scenario. Furthermore, DozyAP can be improved by leveraging existing literature, e.g., by traffic shaping [33,68] and sleeping in short intervals [55].

Sleep scheduling. Sleep/wake scheduling has been extensively studied in Bluetooth domain, e.g., [35, 54] and sensor network domain, e.g., [80, 89]. However, those approaches usually focus on MAC layer design, resulting in a new MAC protocol, and often require time synchronization. DozyAP employs a simple application-level protocol to coordinate the sleep schedule of a softAP with its client, without requiring time synchronization or any modifications on existing IEEE 802.11 protocol. Thus, DozyAP is easy to deploy on existing smartphones.

Dedicated Wi-Fi tethering devices. MiFi [60] is a dedicated mobile Wi-Fi hotspot device. However, such a device also stays in a high power state even without any ongoing traffic. We measured a Huawei E5830 MiFi device and found the average power consumption was as high as 420 mW in idle case. We believe MiFi devices can benefit from DozyAP design if they are programmable.

4.8 Conclusion

In this chapter we have studied the power efficiency of Wi-Fi tethering. We show that Wi-Fi tethering on existing smartphones is power hungry and wastes energy unnecessarily, but there are many opportunities to save power by putting a mobile softAP to sleep. We propose DozyAP system to improve the power efficiency of Wi-Fi tethering. DozyAP employs a lightweight yet reliable sleep request-response protocol for a mobile softAP to coordinate its sleep schedule with its clients without requiring tight time synchronization. Based on our findings on the traffic patterns of typical applications used in Wi-Fi tethering, we design a two-stage adaptive sleep algorithm to allow a mobile softAP to automatically adapt to the ongoing traffic load for the best power saving. We have implemented DozyAP system on commercial smartphones. Experimental results demonstrate that DozyAP is able to significantly reduce the power consumption of Wi-Fi tethering without impairing the user experience.

5 Group Localization via Gestures

In addition to the security and energy-saving discussed in the previous chapters, location-awareness is desired more than ever as this world has become increasingly mobile. However, current localization techniques are inadequate to meet different demands for location such as fine-resolution, low-cost and scalability. To shed light upon the general research that explores more comprehensive location-aware techniques for mobile systems, this chapter describes a novel group localization scheme that can be used for mobile user to locate others in ad hoc wireless networks.

5.1 Introduction

Consider the following scenario. A group of people at a conference or a workshop come together in a room for a meeting. They may not know each other in advance, so have no knowledge about each other's name. They would like, however, to share information or exchange electronic documents during the meeting. They will use their electronic devices (smartphones or laptops) to quickly set up a local network and then localize each person on a simple map. This location information is extremely useful for people to identify each other. If the involved people choose to remain anonymous, they can use an intuitive gesture to transfer information between each other. For example, when two people would like to share files, they can use *swipe-to-share*, in which users can directly send files by swiping their phones towards the receiver. This type of technology can be potentially applied to other scenarios as well. Game players use gestures to communicate with each other in recently emerging phone-to-phone games [97]. People or parties exchange electronic documents during contract negotiation.

It is our expectation that this type of group localization should meet the following criteria: (1) Scalability. The solution should be lightweight so that it can scale to a reasonable sized group. (2) Accuracy. While exact positioning is not required, the localization error should be reasonably small so that group members can be identified. (3) Security. Only devices within certain range can be included in the group. The design should be resilient to adversary attacks.

A large body of research has been dedicated to localization without Global Positioning System (GPS) for various purposes over decades. Most work relies on infrastructure support such as Wi-Fi access points (APs) and cellular towers, where the location of each device is either assigned by central server or self-computed and shared with other devices via network communication. However, the infrastructure support may not be available in every phone-to-phone scenario, thus limiting the generality of such approaches. On the other hand, if the location is self-computed from such as RSSI, RF signature, or magnetic fields, malicious users may cheat on their locations to launch attacks. Moreover, these techniques typically cannot achieve satisfactory accuracy. Other localization approaches demand special hardware design, complex signal processing, or access of very low layer information. Due to the constraints, those approaches can not be easily applied to smartphones. Recent work proposes some acoustic ranging based techniques that can be used for smartphones with speakers/microphones. However, they have some practical limitations. In BeepBeep [66], every device beeps at least once to measure the distance between each other and then calculate the location of each device. The limitations of this approach are: (1) at least three devices are needed to perform localization, otherwise only distance can be derived; (2) all devices have to collaboratively beep, thus making the approach not scale well with the number of devices; and (3) this approach requires that every user agrees on the same beeping order and is synchronized well to avoid the collision of beeping simultaneously. Based on BeepBeep, another localization algorithm that can be used between two phones is proposed in [71]. This approach requires time synchronization and still does not scale well with multiple devices.

In this chapter, we propose SG-LOC, a system to efficiently group and locate mobile devices. The localization process of SG-LOC cleverly strengthens BeepBeep using on-broad sensors and Multilateration algorithms. In our system, the beeps are only emitted by a single user who is responsible for setting up a local network. During the localization, a user is requested to hold her phone and draw a simple gesture. After that, the locations of other devices are calculated and shared with other users when they join the network. Our approach does not rely on infrastructure support for communication. Furthermore, the localization process scales well with the number of devices. We have implemented SG-LOC on commercial smartphones. The experimental results show that our system can achieve centimeter-level accuracy.

To the best of our knowledge, this is the first use of on-broad sensors and audio to locate mobile devices. The main contributions of this work are:

- We design SG-LOC, a system to efficiently group and locate mobile users in proximity. The localization process cleverly leverage on-broad sensors and multilateration algorithms without any infrastructure support.

- We characterize the challenges of implementing SG-LOC, namely significant drift in integration of inertial measurement unit (IMU) sensors, signal detection in environment with highly self-correlated ambient noise, sampling rate drift between different devices, best moving strategy for drawing gestures.

- SG-LOC operates in the application layer without relying on any modification to the OS. We have implemented SG-LOC on commercial smartphones and performed extensive evaluation.

The rest of the chapter is organized as follows. The related work is presented in Section 5.2. The main idea and system architecture of SG-LOC, followed by the algorithm for each component, are described in Section 5.3. We present our implementation in Section 5.4 and evaluation in Section 5.5. Limitations and the future work are discussed in Section 5.6. Finally, we conclude in Section 5.7.

5.2 Related Work

Localization without GPS has been studied over two decades. Most of techniques can be broadly classified into two categories based on the type of media: *Radio Frequency (RF) based techniques* and *Acoustic techniques*. We name some related works in the following.

RF based techniques. In these techniques, location is determined by measuring the radio signals sent from Wi-Fi APs or cellular towers. Among them, received signal strength (RSS) is commonly leveraged, since it can be easily measured by commercial wireless cards. By profiling RSS fingerprints for each location, localization is performed by finding a location with the matched fingerprint [21,63,92,94]. Besides that, FM radio [30] and channel responses from multiple OFDM subcarriers [73] are recently proposed as signatures. Different from signature-based approaches, several techniques exist for deriving range, angle and proximity information from radio signals, and then positions can be inferred by applying geometric algorithms. Time-of-arrival (TOA) systems such as [95] determine the distance between devices by measuring RF propagation delays. Time-difference-of-arrival (TDOA) systems such as [51] rely on the signal difference in arrival time and phase on time-synchronized devices to determine range. Angle-of-arrival (AOA) systems [34] utilize the directions from which a signal is received to derive positions. Through measuring the RSS of RF signals, the location of devices can be also determined by employing a radio propagation model [22, 32, 38]. These approaches do not provide provisions to accurately locate nearby mobile users in any circumstance, since they typically need profiling in advance, special hardware design, or only provide coarse-grained precision (e.g., room-level).

Acoustic techniques. Acoustic techniques can measure the range more precisely, owing to its relatively slow speed compared with RF signal. Hence, most acoustic localization schemes leverage range-based approaches. Many systems such as [42, 43, 69] adopt custom hardware to measures the time-of-flight of modulated ultrasonic signals to estimate the range between devices. The ENSBox system [37] leverages microphone array to obtain orientation information for localization. These approaches cannot be applied to mobile phones without additional hardware. The BeepBeep system [66] designed to work with ordinary mobile devices with speaker/microphone introduces a novel way to measure the range based on the elapsed time between two time-of-arrival (ETOA) of two audio tones. Based on BeepBeep, the work [71] uses multiple speakers and microphones to perform phone-to-phone localization in 3D space. SwordFight [97] improves BeepBeep by supporting fast and continuous phone-to-phone ranging. Different from those approaches, only a single device emits audio tones in our work, thus eliminating the requirement of time synchronization and significantly improve the scalability. Recent work [52] proposes another acoustic TDOA-based ranging technique for mobile phone self-localization with infrastructure support. In addition, acoustic fingerprint is also used for indoor localization such as the work [77].

Miscellaneous. There also has been research focused on hybrid techniques of both RF based and acoustic localization. WALRUS [25] can achieve room-level localization in office environment by broadcasting the identity of the room through sound and Wi-Fi channels. Centaur [61] improves the resolution of localization by acoustic ranging plus Bayesian inference. Acoustic ranging techniques are also leveraged to detect driver phone use [91], and pair indented devices by a pointing gesture [67]. Other related works in the context expect for localization is to leverage IMU sensors for the movement recognition. The techniques proposed in the work [18] can recognize human handwriting using phones. Methods described in the work [56] can recognize human activities. Those algorithms relies on IMU sensors to extract the features of the movement, while our targeted problem is more challenging that demands measuring the precise displacement of the movement.

5.3 System Design and Algorithm

In this section, we introduce *SG-LOC*, a system uses acoustic signals to group and locate mobile phone users in proximity simultaneously. In our system, a user who is responsible for setting up a local network is requested to move her phone through a simple gesture to locate other users. After the network is set up, the relative location of each user is displayed on every user's screen. Next, we present the key idea of our system followed by the architecture and its components.

5.3.1 Overview

In the rest of this chapter, the user who initiates the process of localization is called the sender and other users are called receivers. The main idea of our localization scheme is illustrated in Fig. 5.1 for the case of one sender and one receiver. To locate the receiver, the sender will move her phone through a simple gesture, say from Loc1 to Loc2 to Loc3 and then back to Loc1. During the movement, several audio tones are emitted in different locations. Suppose the receiver is located at

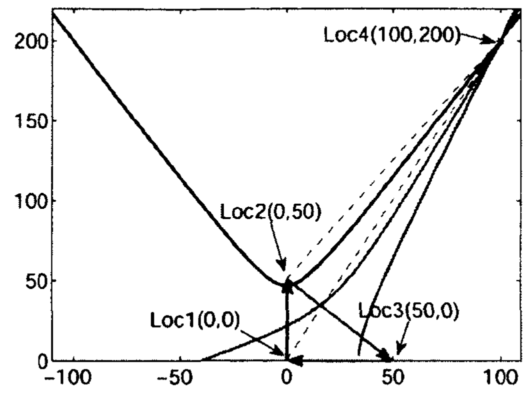
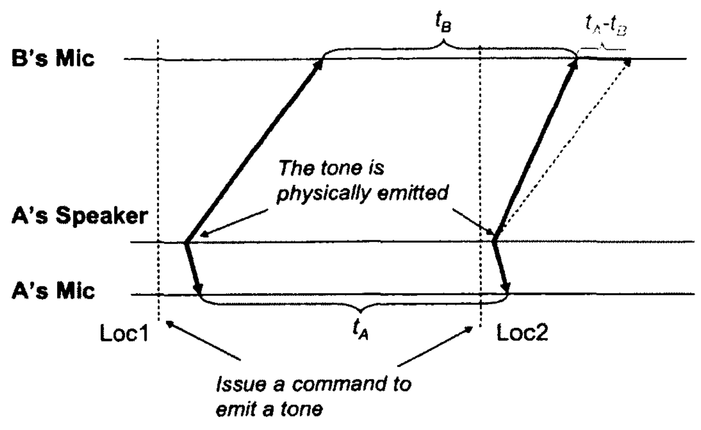


Figure 5.1: Illustration of the algorithm where the sender's phone moves from Loc1 to Loc2 to Loc3 and back to Loc1. The receiver's phone is stationary at Loc4.

Loc4. Due to the spatial change of sender's location between two consecutive tones, the time interval measured by the sender is different from that measured by the receiver. Fig. 5.2 depicts such a difference for the first segment of the movement (i.e., from Loc1 to Loc2), where t_A and t_B are the time intervals measured by the sender and the receiver respectively. Because the Loc2 is closer to the receiver, t_B is slightly smaller than t_A. With $t_A - t_B$ and the speed of sound, we can compute the distance difference dd using the following equation:

$$dd = d_{14} - d_{24} = c \cdot (t_A - t_B),$$

where d_{14} is defined as the distance between Loc1 and Loc4 and d_{24} refers to the distance between Loc2 and Loc4. Given dd, a *hyperbolic curve* is determined, which indicates the possible locations of the receiver (i.e., every point on the curve will incur the same distance difference). Any pair of tones can determine such a curve, so that the receiver is finally located at the intersection point. Leveraging this idea, our system can position multiple receivers on a 2D plane by just three beeps. The remaining questions are how to obtain the coordinates of each point (i.e., Loc1, Loc2 and Loc3) and how to measure the distance difference. We utilize



t_B
t_A-t_B
B's Mic
The tone is physically emitted
A's Speaker
A's Mic
Loc1
t_A
Loc2
Issue a command to emit a tone

Figure 5.2: Illustration of the difference of beep interval between the sender and the receiver.

IMU sensors to address the first issue, and use speakers/microphones for the second issue. In the following sections, we elaborate the system architecture followed by each component.

5.3.2 Architecture

Fig. 5.3 depicts the system architecture of SG-LOC. In our system, only the sender emits beeps which are triggered by motion sensors. The rule is as follows: the first beep is triggered by the initial movement. That location is referred to as the origin (0,0). During the movement, each pause will trigger a beep sound. When the phone returns to the original location, the last beep is played. Using this rule, each beep is emitted at the ending point of every segment. As a result, the Doppler effect that harms the accuracy of any acoustic localization scheme is mitigated naturally. The reason why we need the sender return back to the origin is to compensate various drifts that we elaborate later.

Specifically, the *movement tracking algorithm* ① and *beep detection algorithm* ② on the sender side are used to track the trail of the movement and measure the beep intervals respectively. On the receiver side, the beep intervals derived from the microphone are transmitted to the sender at the time of association. Based on

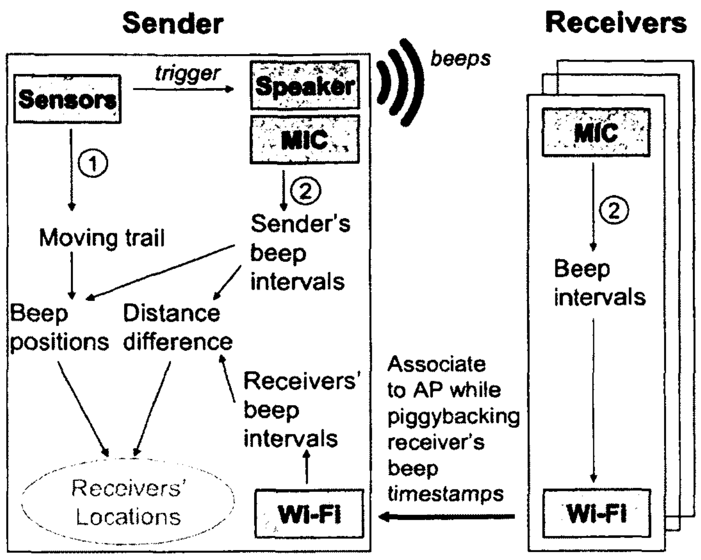


Figure 5.3: System architecture of SG-LOC

the movement trail and the beep intervals, the sender can calculate the coordinates when emitting a beep. Based on the beep intervals on both sender and receiver sides, the sender can derive the distance difference mentioned in Section 5.3.1. All of these are finally used to compute the location of receivers via our positioning algorithm. In next sections, each component is presented in detail.

5.3.3 Positioning Algorithm

Since three beeps are minimal requirement to position a device in a 2D plane, we first present the positioning algorithm for three beeps and then extend it to more than three beeps.

Positioning algorithm for three beeps. Suppose a receiver is located at the coordinates (x, y). The coordinates of three beeps are $(0, 0)$, (x_1, y_1), and (x_2, y_2) respectively. The distance difference derived from beep intervals on both the sender and receiver sides are denoted as dd_1 and dd_2. Thus, we have the following equations:

$$dd_1 = \sqrt{(x-x_1)^2+(y-y_1)^2} - \sqrt{x^2+y^2}$$
$$dd_2 = \sqrt{(x-x_2)^2+(y-y_2)^2} - \sqrt{x^2+y^2} \tag{5.1}$$

Combining the above two equations produces a linear equation for x and y in terms of $Ax+By=C$, where A, B and C are all constant variables given (x_1, y_1), (x_2, y_2), dd_1 and dd_2. Now either x or y can be expressed by the other variable and substituted back into Eq. 5.1 to derive the closed-form for x and y using (x_1, y_1), (x_2, y_2), dd_1 and dd_2.

The *triangle inequality* states that the difference of lengths of any two sides must be less than the length of remaining side. That means the absolute values of dd_1 and dd_2 must be less than the moving lengths $\sqrt{x_1^2+y_1^2}$ and $\sqrt{x_2^2+y_2^2}$ respectively. In practice, due to measurement errors dd_i may be slightly larger the $\sqrt{x_i^2+y_i^2}$ or less than $-\sqrt{x_i^2+y_i^2}$ where $i \in 1, 2$. In both cases, we cannot solve the equations. Instead, if in the former case, the receivers are estimated somewhere in the moving direction. Otherwise, the receivers are deemed in the opposite of the moving direction.

Solving Eq. 5.1 may produce one, two or none solutions in practice. If two solutions exist, additional information is needed to select one point. For example, in a meeting room where all people are sitting around a table and every people is in front of the others, the solution indicating that the receiver in behind should be ignored. For the none solution case, extra beeps are needed.

Positioning algorithm for more than three beeps. With more than three beeps, the happenings of abnormal cases can be reduced and the positioning accuracy can be improved. Let the coordinates of the first beep be $(0, 0)$ and (x_i, y_i) $i = 1, 2, 3, \cdots$ for other beeps. We have an equation for each delta distance as follows:

$$dd_i = \sqrt{(x-x_i)^2 + (y-y_i)^2} - \sqrt{x^2+y^2}.$$

Similar to Eq. 5.1, combing every dd_i and dd_1 yields a linear equation in two variables x and y. Suppose there are n beeps. All of $n-2$ linear equations can be expressed in the following matrix form

$$\begin{bmatrix} A_1 & B_1 \\ A_2 & B_2 \\ \cdots & \\ A_{n-2} & B_{n-2} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} C_1 \\ C_2 \\ \cdots \\ C_{n-2} \end{bmatrix}.$$

As long as $n \geq 4$, variables x and y can be solved without substituting them back to the equation of dd_i. The least squares method can be used to find an approximate solution to the overdetermined system. The point derived by such a method has the minimal distance to all the curves. However, according to our experience, least squares method does not work well in practice, since some curves are more noisy than others due to measurement errors. Including these curves for calculation may degrade the overall accuracy. Our method is to first select a unique set of three beeps from n beeps. The total number of sets is equal to $\binom{n}{3}$. For each set, the positioning algorithm for three beeps is applied to extract possible solutions. The direction of those points to the origin is then calculated. The majority of points within a certain angle (10° is used in current implantation) are kept and others are filtered out as outliers. Finally, the geometric centroid of all remaining points is determined as the location of the target. Although the accuracy of localization may be slightly deteriorated when a very accurate solution is averaged by several less accurate solutions, this method can output a good solution for general cases.

5.3.4 Movement tracking algorithm

Our system uses IMU sensors namely accelerometer and gyroscope to track the movement of the phone. These motion sensors on smartphones are typically low cost Micro-Electro-Mechanical Systems (MEMS) that are not specifically used to track the precise movement. Therefore, the accurate displacement measurement of the phone is challenging. The main measurement errors come from the following sources. First, accelerometer and gyroscope are expected to report the readings on axes of phone's frame. However, if the internal measurement units are slightly misaligned with the phone frame during fabrication, the reported readings are drifted. The second error source is from the gravity. If the screen plate of the phone is not orthogonal to the gravity direction, both x-axis and y-axis readings may be added by a small component of the gravity. For example, if the screen plate is skewed about 1 degree, the gravity provides $0.17m/s^2$ (i.e., $9.8m/s^2 * sin(pi/180)$) component to x-y axis plate, thus the phone appears to move 8.5cm distance after 1s even though it is actually stationary. Hence, how to remove the gravity correctly is critical to the displacement measurement. Third, owing to the Brownian motion, white noise is generated by thermal agitation due to the random charge carrier motion inside the sensors, resulting in a noisy signal sensor output. Last, during the movement phone's orientation may change, while each sensor reading is in phone's instantaneous coordinate system. For example, before moving the phone's y axis is towards the north. After moving once, the phone's y axis is towards the east. The current y axis readings are actually the x axis readings before, and the x axis readings now are the negative y axis readings before. Without transforming all sensor readings into a reference coordinate system, the estimated positions are not useful.

To derive displacement, we double integrate the readings from accelerometer. The first integration transforms the acceleration to velocity, and the second integration then converts the velocity to the displacement. Since the sensor readings are generated at a certain sampling rate, we use the *trapezoidal rule* to numerically approximate the integral between two samples. Suppose a sample is generated at timestamp t_1 followed by another sample at t_2, and the values of each sample are denoted by $f(t_1)$ and $f(t_2)$ respectively. The trapezoidal rule works by approximating the region under the function f between t_1 and t_2 by calculating:

$$\int_{t_1}^{t_2} f(x)dx \approx (t_2 - t_1)\frac{f(t_1) + f(t_2)}{2}.$$

As mentioned above, there exist many errors in sensor readings. If we directly apply double integration on raw readings, these errors will accumulated by each integration and lead to significant drift. To address this problem, we propose the following approaches to reduce the drift.

Motion detection. The integration will accumulate the errors into drift. Typically, the longer the period of integration, the larger the drift. To reduce the drift, it is important to decrease the integration period. The first step is to detect when the phone is moving and when it is not, and only integrate on the period when the motion is detected. To detect the motion, we separate all samples into bins with each size equal to 10 samples and compute the standard deviation of each bin. If the deviation in a bin is greater than a pre-defined threshold, the first sample in such a bin is conservatively regarded as the starting of a movement. Given a motion is detected, if the deviations in consecutive two bins are less than the threshold, the stop of that movement is detected.

Rotation transformation. Rotation transformation seeks to transform the acceleration from phone's frame to a global reference frame. Since the effect of gravity is consistent, it can be eliminated by velocity compensation described later. To perform rotation transformation, we first need to compute the rotation matrix. The gyroscope measures the instantaneous angular speed around phone's x, y, and z axis. Given the initial phone's frame is the reference frame, the output of the gyroscope is integrated over time to calculate the angle rotation from sampling instance to the initial. Again, we only integrate the readings when movement is detected. Suppose at timestamp t_1, the angle changes on x, y, z axis are roll (ϕ), pitch (θ), and yaw (ψ) respectively. The rotation matrix is calculated as follows,

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\phi & -sin\phi \\ 0 & sin\phi & cos\phi \end{bmatrix} \begin{bmatrix} cos\theta & 0 & sin\theta \\ 0 & 1 & 0 \\ -sin\theta & 0 & cos\theta \end{bmatrix} \begin{bmatrix} cos\psi & -sin\psi & 0 \\ sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

The product of the rotation matrix and the accelerometer readings in phone's frame yields the transformed acceleration.

Velocity compensation. When users hold their phones to draw a triangle, it is natural for them to stop at three corners. During the stop, the phone should be stationary and its speed is zero. However, the acceleration residues caused by the gravity and the misalignment of internal sensors typically are integrated to a non-zero velocity at a stop. If the velocity is not compensated, the integrated displacement will drift significantly. The method to compensate the velocity is as follows. First, we detect three segment movements using the method just mentioned before. At the end of each segment, if the integrated velocity is not zero, all velocities integrated within such a segment is adjusted to the following value:

$$v'(t_j) = v(t_j) - v(t_k)\frac{t_j - t_i}{t_k - t_i},\ i \in [i, k]$$

where $v'(t_j)$ denotes the adjusted velocity at t_j, $v_(t_j)$ refer to the originally integrated velocity at t_j, t_i and t_k represent the starting and ending time of a segment. Using this method, constant gravity and sensor misalignment offset can be eliminated at the same time.

Relative displacement compensation. In our scheme, the phone will return to the original point after drawing a triangle. Thus, the relative displacement between the starting point and ending point should be zero. However, due to the errors mentioned at the beginning of this section, the integrated displacement may not be zero. To have an accurate displacement measurement, we need to compensate the integrated displacement over time. Slightly different from the velocity compensation that is applied to each segment, the relative displacement compensation is applied to the whole triangle once. First, we detect the beginning of the first segment t_1, the end of the last segment t_n, and all stationary duration $st(t_i)$ before a timestamp t_i. The displacement is adjusted as follows:

$$d'(t_i) = d(t_i) - d(t_n)\frac{t_i - t_1 - st(t_i)}{t_n - t_1 - st(t_n)}.$$

5.3.5 Beep detection algorithm

Due to the uncertainty delay between the instance when a command is issued to emit a tone and the instance when the tone is physically emitted by the hardware, the actual interval between two beeps on the sender side cannot be accurately determined by the beep issuing time in the software. To address this problem, we adopt the method similar to works [66], where the sender needs to record audio waves from its microphone when beeping. Since the distance between the sender's speaker and its microphone never changes, the time interval between two beeps captured by the microphone is exactly the same as they are physically emitted by the speaker.

Furthermore, we can detect the starting points of beeping by processing the audio files recorded from the microphone, and calculate the time interval by counting how many samples between two beeps. Compared with directly using software timestamps to calculate the interval, converting the number of samples to time can yield more accurate results. That is because the sampling is typically performed by hardware. Each sample is performed at a specific rate (e.g., 44.1 kHz). The number of samples divided by the sampling rate yields the time duration. This method can cancel out the uncertainty delay between the instances when the beep is detected by the hardware and seen by the software. Furthermore, this method does not require clock synchronization between devices. It is practical for implementation. As shown in Fig. 5.2, at Loc1 and Loc2, the sender A emits two beeps. Both A and the receiver B record audio from their microphones and obtain TDOA: t_A and t_B. Then beep interval is calculated as $t_A - t_B$.

5.3.5.1 FFT-based beep detection

For better performance, the beep signal should be designed carefully to cope with the following issues. First, the sound signal will be attenuated and distorted by the communication channel, and negatively affected by the environment noise. Thus, the signal should be designed to have a good Signal-to-Noise Ratio (SNR) at the receiver. Second, the signal should have a better resistance to *multi-path* and *non-line-of-sight (NLOS)* effects. Due to these effects, the first arrival signal at the receiver may not have the largest energy. It imposes difficulties to detect the arrival signal and determine the DTDOA accurately. Based on the above considerations, we choose a linear chirp waveforms.

The waveform detection is performed by the matched filtering process, where the emitted signal is correlated with received signals to determine when a beep is present in the received signals. Let $\{u_i\}$ $(i = 1, 2, \cdots, n)$ denote the signal sequence recorded from the microphone, and $\{v_i\}$ $(i = 1, 2, \cdots, m)$ represent the emitted signal (i.e., a beep). In matched filtering, a sliding window with the length equal to m moves from the beginning to the end of $\{u_i\}$ sample by sample. The sample correlation coefficient r is computed as follows:

$$r = \frac{\sum_{i=1}^{m}(u_i - \overline{u})(v_i - \overline{v})}{\sqrt{\sum_{i=1}^{n}(u_i - \overline{u})^2}\sqrt{\sum_{i=1}^{n}(v_i - \overline{v})^2}},$$

where $\overline{u}$ and $\overline{v}$ are sample means of the sliding window and $\{v_i\}$ respectively. The value of r may vary for each sample. A large r means there is a high similarity between two sequences. As a result, any sample with a much larger coefficient than its nearby samples is identified as the arrival of a beep. The number of samples between two beeps divided by the sampling rate infers the beep interval.

However, the calculation of correlation coefficient is computationally expensive especially when n is large. Some work [61,66] offloads the computation to a powerful cloud server or leverage the parallel GPU hardware to accelerate the process. Such approaches work but at the expense of extra communication overhead or power consumption. In recent work, Qiu et al [71] propose to use an energy threshold to reduce the search space in the sample sequence. However, setting such a threshold is challenging because the ambient noise may be large compared to the emitted sound in some scenarios. If the threshold is set too low, the computation overhead is still considerable. By contrast, if the threshold is set too high, not all beeps can be captured. To overcome this problem, Zhang et al. [97] uses auto-correlation to estimate a rough position of a beep and then apply cross-correlation to identify the exact position. Auto-correlation characterizes the self similarity of a sequence. In their approach, a beep is composed of two same sequences: one followed by the other. If a sequence is found most similar to its half-lag (-shift) sequence, a beep is detected. Auto-correlation can be computed fast but inherently has fatten peak, so it is difficult to detect the position of a beep accurately. That is why cross-correlation is used in the second stage. However, this method cannot be applied to the scenario with highly self-correlated ambient noise, such as AC.

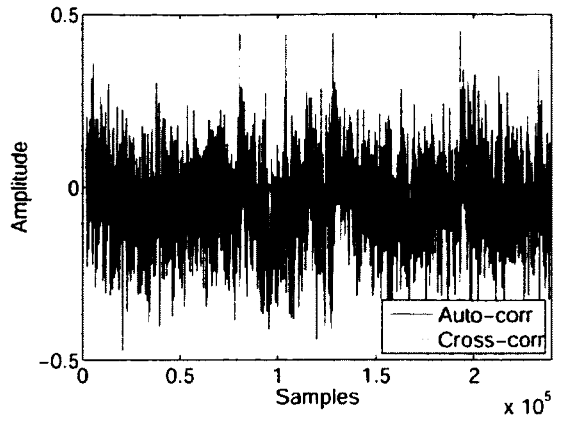


Figure 5.4: Beep detection by auto-correlation and cross-correlation, where sampling rate is 44.1kHz

Fig. 5.4 shows results of both auto-correlation and cross-correlation of recorded samples in our office. It is seen that the auto-correlating hardly detect beeps due to the background noise.

In this work, we propose a fast Fourier transform (FFT) based filter to reduce the unnecessary search space. First, the emitted chirp sound is converted to frequency domain by FFT. Next, the received sequence is divided into equal blocks with the same size as the chirp. In each block, FFT is computed to find whether a similar spanning frequency is detected. If so, a beep is found. In this block and its previous block, we then use cross-correlation to derive the accurate position. Otherwise, we skip this block to reduce computation overhead. It should be noticed that the previous block must be considered, because a beep may cross two consecutive blocks in received signals. Without taking the previous block into account, we may have a chance to miss a beep. Suppose N beeps are emitted each with length equal to m and the received signal has n samples in total. To detect the position of all beeps, our approach needs about $\Theta(n \log m)$ time to process FFT plus $\Theta(m^2)$ time to calculate cross-correlation. Later in the evaluation section, we present experiment results to confirm the efficiency of this method.

5.3.5.2 Sampling rate drift

In practice, different smartphones may have a slight drift on a given sampling rate. In other words, during a certain period of time, two phones are very possible to collect different number of samples even though the sampling rate is the same. According to our experiments, this drift often occurs on different brand of phones. For examples, two phones: a Samsung Galaxy Nexus and a HTC evo 3D, are used for test. The nexus emitted two beeps separated by 3s. Both phones were stationary and recorded samples from their microphones. We found that the number of samples recorded by evo was always less than that of nexus by 14 samples. In that case, the measured distance difference is inaccurate. To our best knowledge, this is the first work that pinpoints the impact of sampling rate drift on acoustic ranging systems.

To cope with this problem, we request users to move their phones back to the original location. The first beep and the last beep are then emitted at the same location. With this assumption, we can estimate the speed of drift and compensate the measured beep intervals. According to our extensive experiments, we found the drift rate is constant over time. Hence, a linear model is good enough to correct the intervals.

5.4 Impact of measurement errors

In this section, we discuss the impact of possible measurement errors on the localization accuracy and then present the guidelines for our system design. Basically, there are two sources of errors in our system: *distance difference errors* due to the acoustic subsystem and *displacement measurement errors* owing to the IMU sensors.

5.4.1 Distance difference errors

In our acoustic subsystem, some factors may cause inaccurately measuring the distance changes between sender and receiver due to sender's movement. First, since we use acoustic signals to detect the distance change, the resolution of detecting a tiny change in distance is determined by the sampling rate of speakers/microphones. For example, recent commercial sound cards support maximum 44.1kHz sampling rate, so that the minimal interval between consecutive samples is 22.7μs. With the speed of sound equal to 341m/s, a sample difference is translated to $0.8cm$ changes in distance. In that case, the measured distance changes can be only the multiples of $0.8cm$, resulting in resolution-related errors. Second, the acoustic signal will be distorted on both phase and frequency when received at the receiver due to the multi-path effect and the ambient noise during propagation. Thus, the cross-correlation method may not detect the precise time when the beep arrives. As a result, the distance difference derived by both sender and receivers may include errors. To analyze the impact of these errors, we assume there exists a bound Δr. It means that if the exact distance difference is r, the measured value r' should fall into the range $[r - \Delta, r + \Delta]$. According to the results from previous research and the verification by our experiments, the standard deviation of distance difference errors are within $1cm$ in practice. Therefore, we set $\Delta r = 1$.

Consider on a 2D plane the sender moves along a segment and beeps at two spots. If the distance change is measured without any error, a curve consisting of possible locations should pass through the exact receiver. However, measurement errors make the curve depart from the actual locations. A lower bound and a upper bound of such a departure are determined by $r - \Delta r$ and $r + \Delta r$. Fig. 5.5 illustrates an example, where the target's phone is located at the coordinates $(0, 500)$, while the sender's phone is moved from $(0, 0)$ to $(21, 21)$ and beep at each point. The

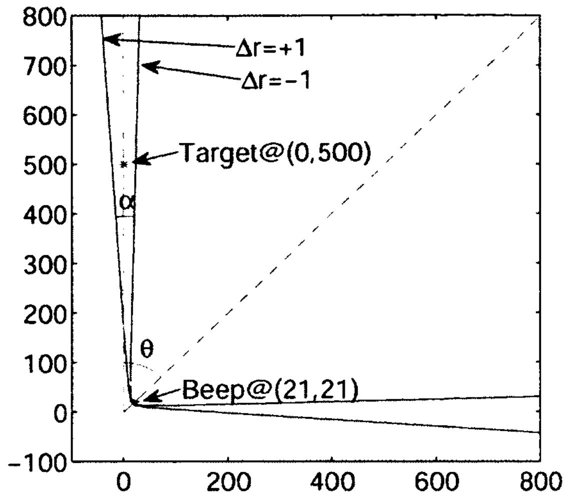


Figure 5.5: Possible sector of the target device determined by a single movement, where the maximum measurement errors of distance change is $\pm 1cm$

angle between the target's direction and the direction of movement is defined as θ. The target may exist at any point within the sector embraced by both lower bound and upper bound curves. We define α as the angle of such a sector. A smaller α is better with respect to the localization accuracy. This example is used to demonstrate the impact of different direction of movement (i.e., θ) on the localization accuracy (i.e., α). Note that even though different numbers rather than $(0, 500)$, $(21, 21)$, and $\pm 1cm$ are used, the conclusion is still similar.

Fig. 5.6 depicts the relation between θ and α, where the value of θ ranges from -180° to 180°. The positive angles represent the clockwise rotations, whereas the negative angles refer to the counter-clockwise rotations. We see that in the case that the sender's device is moved towards or opposite to the direction of target, the corresponding α is largest, i.e., the lowest localization accuracy. However, in the other case that the direction of movement is perpendicular to the direction of target, the smallest α is achieved. Therefore, the first guideline for movements is conducted that *it is better to move perpendicular than parallel to the direction of target.*

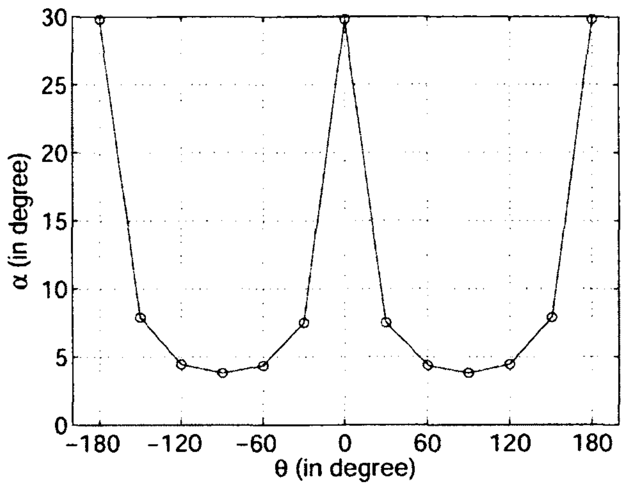


Figure 5.6: Relation between θ and α, where θ defines the angle of target direction and moving direction and α refers to the sector angle both illustrated in Fig. 5.5

From Fig. 5.6, we find that any negative θ and its corresponding positive θ with the same absolute value have the same α. In fact, they have different capability to locate the target. Among all errors in the sector determined by the lower bound and upper bound curves, if the estimated direction of target is greater than the exact direction of target (i.e., on its right side), we define these errors as positive ones. By contrast, if estimated location of target is on the left side, we define them as negative errors. Fig. 5.7 shows the percentage of positive and negative errors against θ. It is seen that when θ is negative, angle errors are prone to be positive. On the other hand, positive θs are likely to cause negative angle errors. Recall that our scheme demands at least two movements to compute a determined location of target. By overlapping positive errors and negative errors, the final error should be reduced significantly. Therefore, the second guideline for movements is that *if moving more than once, the directions of movements are better to flank the direction of target.*

5.4.2 Displacement errors

We have already discussed the effect of errors in measuring the distance difference on the localization accuracy. In this section, we elaborate the effect of errors

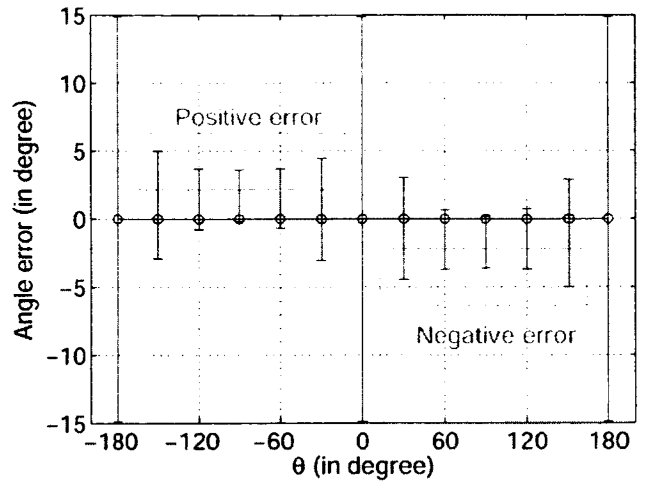


Figure 5.7: Distribution of angle errors against θ

in measuring the displacement of sender's phone. The reasons why we have displacement errors are described before. Suppose the exact movement of sender's phone is from (0,0) to (x_1, y_1). Due to errors, the estimated trail of movement is from (0,0) to (x'_1, y'_1) that may (1) have variant distance, and (2) rotate slightly from the exact direction of movement. In the following, each case is discussed individually. First, we assume that (x_1, y_1) and (x'_1, y'_1) are in the same direction but have different distance $\sqrt{{x'_1}^2 + {y'_1}^2} \neq \sqrt{{x_1}^2 + {y_1}^2}$. The maximum deviation between (x'_1, y'_1) and (x, y) determines an error region. We then vary the direction of movement and examine the area of the error region. The results are quite similar to Fig. 5.6 and Fig. 5.7. It is concluded that moving perpendicularly to the direction of target is better than moving in parallel. Second, we assume the distance is accurately measured but in different orientation $\frac{y'_1}{x'_1} \neq \frac{y_1}{x_1}$. In this case, we find the direction of movement has little effect on the localization accuracy. Combining two cases, the aforementioned guidelines are still valid.

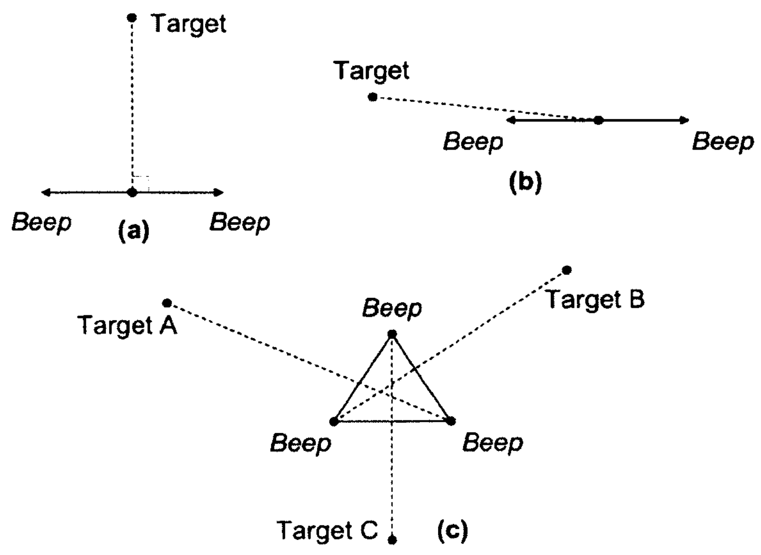


Figure 5.8: Distribution of angle errors against θ

5.4.3 Better moving strategy

How to decide a moving strategy is important because it affects the accuracy of localization. If knowing the location of target in advance, the best strategy for two movements is shown in Fig. 5.8(a) according to the guidelines. However, that strategy may result in a poor estimation for a target whose location is approximately parallel to the direction of movement like Fig. 5.8(b). Therefore, without knowledge of target's location, an equilateral triangle is the best moving strategy to achieve a better performance for any target on average. That is because no matter where the target is located, we can find two sides that can embrace the direction of target (see Fig. 5.8).

In practice, our scheme is not very sensitive to the shape of triangle. People are free to draw any shape of triangle without sticking to a perfect equilateral triangle. Instead of the shape, how people draw the triangle such as the speed of movement is more critical to the accuracy. It also should be noticed that to improve the robust of our localization algorithm, five beeps are emitted along a triangle gesture. We design a new moving strategy to draw a triangle: starting from the middle of the bottom side to the left corner to up corner to right corner and then back to the origin. That strategy combines the benefits of both line and triangle gestures. The detail is described in the evaluation section.

5.5 Security Enhancement

In this chapter, we consider two types of attacks. In the first attack, malicious users may launch *Denial-of-service (DoS) attacks* to disrupt the process of localization. Here, we assume adversaries can eavesdrop legitimate beeps but can hardly jam those beeps. First, since each beep is very short and randomly emitted by users, predicting the precise timing for jamming is very difficult. Second, it is easy to be detected if adversaries jam the sound channel for a long time. Instead, a very effective DoS attack is for the adversaries to generate fake beeps to impersonate the sender and prevent other users from correctly calculating beep intervals. To mitigate this attack, our security-enhanced system leverage a hash chain to ensure that users can derive beep intervals based on legitimate beeps.

Before emitting beeps, the sender first generates a random number R and calculates a hash chain of R. Let $h^n(R)$ denote the output of hashing R n times. Initially, the head of the hash chain (say, $h^{100}(R)$) is broadcasted to users via beacons. When the user emits a beep, a linear chirp sound followed by an audio tone encoded with the next value in the chain (say, $h^{99}(R)$, $h^{98}(R)$ and so forth) is played. After locating every beep in samples, the receiver can decode a hash value for that beep. To verify whether the ith beep is valid, the receiver computes the hashing of the received value one more time (e.g., $h(h^{98}(R))$. If the output matches the value for the $i-1$th beep (say $h^{99}(R)$), the beep is accepted otherwise rejected. Each time the hash value is different, and thus cannot be guessed or duplicated by an attacker.

Without broadcasting the head of pre-calculated hash chain, our scheme still works but relies on receivers to supply all hash chains (one by the sender and others by adversaries) to the sender for authentication. In this case, all hash chains of previous beeps are stored. For a new beep, the receiver computes the hashing one more time and tries to match with all stored chains. If matched, such a beep is added to an existing hash chain. Otherwise, a new chain is created. In addition, any beep with duplicated hash value is directly discarded. At the end, the beep intervals for all chains are reported. The sender only accepts the intervals belonging to its own chain.

For the second attack, we consider *spoofing attacks* where malicious users who are not in proximity of benign users (e.g., outside the room) try to join the group. Without physically staying closer to users, the adversaries cannot hear all beeps and report valid beep intervals when associating to the network. The user who is responsible for setting up the network can thereby exclude these adversaries. However, more sophisticated adversaries can still join the network by by sending the forged intervals or replaying intervals reported by benign users. To defend against this attack, each user has to prove it received all beeps. The details are described as follows.

As mentioned above, each beep is identified by a unique hash value. Suppose m beeps are emitted by the sender and their hash values are $v_1, v_2, \cdots, v_m$. All these beeps can be found by checking whether $v_1 = h(v_2)$, $v_2 = h(v_3)$ and so forth. To report the intervals between consecutive two beeps, the receiver has to supply a hashed value of $h(v_1 \oplus v_2 \oplus \cdots \oplus v_m, ID)$ to prove all beeps are actually heard. The adversaries who did not hear all beeps cannot calculate $v_1 \oplus v_2 \oplus \cdots \oplus v_m$. Replaying other uses' requests also does not work because the reported intervals are assigned by each user's ID.

5.6 Implementation

We implemented our localization scheme on Google Galaxy Nexus running Android v4.2.1 as a sender and various other Android phones such as Galaxy Nexus (v4.2.1), Nexus S (v4.1.x), HTC 3D EVO (v4.1.x) and Samsung Droid Charger (v2.3.x) as receivers. In the following, we present the details of our implementation.

5.6.1 Beep signal design

Similar to previous research [52, 61, 66], we choose a *linear chirp* as a beep, since the linear chirp offers good pulse compressibility and increased signal-to-noise ratio (SNR), thus making beep signals easy to be detected. For a linear chirp, the signal frequency increases over time. Given the duration of such a chirp and the increasing factor, we can determine the frequency range of a chirp. According to our experience, the frequency range will affect the accuracy of beep detection. The maximum range of spectrum that Galaxy Nexus can support is within [0,22.05k] Hz. To study the effect of different frequency ranges, we first designed a linear chirp with audible frequency range [1k,5k] Hz (humans can hear audio frequencies up to [19k,20k] Hz). Then, we tested another linear chirp with near inaudible frequency range [18k,22k] Hz. After playing these two chirps separately, we found that receivers can detect both signals but with varying levels of accuracy to pinpoint the precise location of the beep in recorded samples. As illustrated in Fig. 5.9, lower frequencies (left figure) typically have smaller sidelobes (marked by left and right circles) compared to the correlation peak (marked by the middle circle). It is easy to distinguish the correlation peak from other sidelobes. However, higher frequencies have large sidelobes, so that once the beep signal is distorted by ambient noise and multipath effect, the correlation peak may be smaller than a sidelobe. Hence, the location of the beep is likely to be falsely identified. To make our system robust

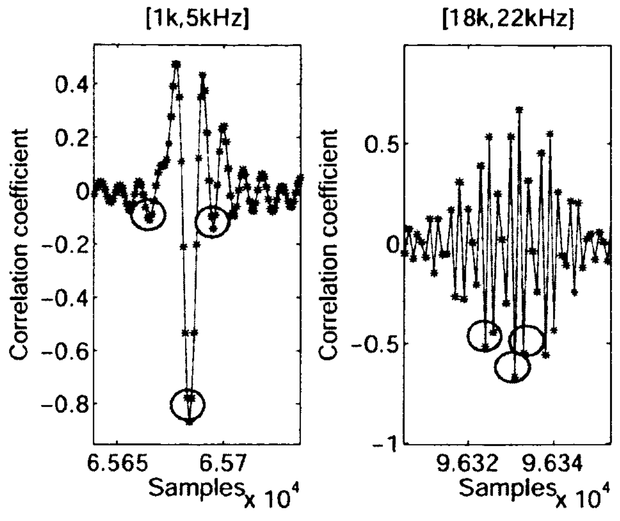


Figure 5.9: Comparison of two frequency ranges of beeps

to dynamic environment, the frequency range of beep is set to [1k,5k] Hz.

For the length of a beep sound, we tested three different durations: 25ms, 50ms and 100ms under 44.1kHz samples rate, and found the length has little effect on the accuracy of beep detection. To reduce the computation overhead, we set the length of beeps to 25ms.

5.6.2 Beep playing design

In our implementation, the time when to play a beep sound is triggered by the movement of the gesture. Before the phone is moved, the first beep is played. Thereafter, once a pause of movement is detected at the ending point of a segment, a beep is emitted. The reasons why we choose to play a beep when phone is relatively stationary are in two aspects. First, Doppler shifts can be mitigated. If a beep is emitted when the phone is moving, the beep may be shifted in frequency domain, making the detection inaccurate. Second, it is easy to find the positions for beep playing from the gesture trail. Since it is difficult to control both IMU sensors and microphone to physically start at the same time, as well as obtain the precise timestamp of each audio sample, translating the beep playing time in samples to beep playing location in the moving trail is not easy. Using this method, each beep is emitted when the phone is stationary, so by looking at the ending point from the moving trail, we can derive the beep playing location without complex time matching.

We use *AudioTrack* class with static mode to play beeps. The buffer size is set to be the minimal for a sample rate of 44.1kHz. We noticed that the beep sound occupies less space than the minimal buffer size. In this case, the beep is not actually sent out until the buffer is full. To reduce the playing lag, we call *AudioTrack.stop* to force playing the sound once the buffer is written. To record waves from microphone, we use *AudioRecord* with an additional shadow buffer to reduce the contention when copying buffer to memory.

5.7 Evaluation

In this section, we use real-world experiments to assess the performance of SG-LOC. We have evaluated our localization algorithms by answering the following questions: 1) What is the accuracy of displacement determination algorithm based on IMU sensors? 2) How severe are the sampling drifts of microphone on commercial smartphones? 3) What is the accuracy of localization algorithms with respect to both angle and distance errors? 4) What is the on average computation time to locate other devices. 5) What is the power consumption of SG-LOC?

In experiments, we use a Galaxy Nexus smartphone with stock Android v.4.2.2 as a sender. Other smartphones including another Galaxy Nexus, HTC EVO 3D, Nexus S, Droid Charger running various Android versions from v2.x to v4.x are used as receivers. Note that the main contribution of this work is the localization scheme, so the network setup is not evaluated.

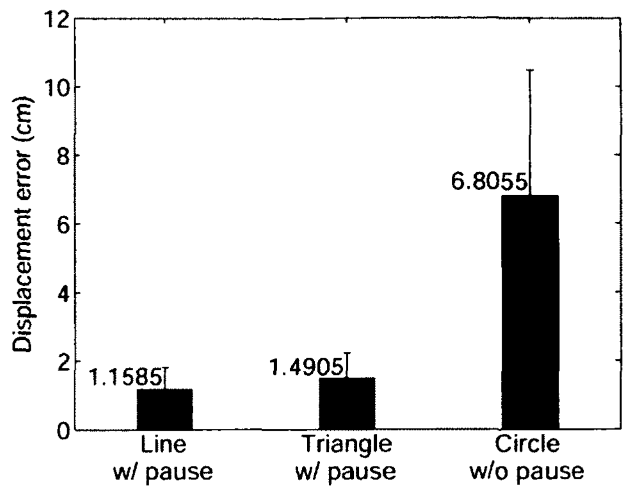


Figure 5.10: Comparison of different gesture with and without pause

5.7.1 Accuracy of IMU sensors

Different gestures and moving strategies have varying effect on the accuracy of our displacement determination algorithm. We investigated the performance under *line*, *triangle* and *circle* gestures with and without pause during the movement. To obtain the ground truth, we first drew specific trails (i.e., line, triangle and circle) as references and asked users to move their phones following the trails.

In the first experiment, a line gesture is tested. Starting from the middle of the line, phone is moved along the line and first towards the left ending point then to the right ending point and then back to the origin. At two ending points, movement will pause for a short time which is decided by users. After the displacement is calculated by our algorithm, we measure the errors between the estimated ending points and the exact points. The experiment is repeated 30 times. The left cylinder in Fig. 5.10 depicts the average results with error bar presenting the standard deviation.

In the second experiment, a triangle gesture is tested. As explained in Section 5.4, drawing a triangle here is slightly different from a regular triangle drawing. The movement first starts from the middle of the bottom side, then towards the left corner, up corner, right corner respectively and then back. At each corner, the movement is also paused. For comparison, we conducted the third experiment with drawing a circle through three reference points without any pause. The middle and right cylinders in Fig. 5.10 show results for each case.

It is seen that line gesture has the smallest estimation error which is slightly less than triangle gesture and significantly less than circle gesture. We conclude that pausing can significantly help the displacement estimation and the gesture costing short time is typically better the gesture costing long time. Therefore, we use simple gestures with natural pause, namely line and triangle for localization.

5.7.2 Validation of sampling drifts

To demonstrate how severe the sampling drifts are, we measured several phones. The results are shown in Table 5.1, where a Galaxy Nexus phone is a sender and other phones are receivers. Six beeps are played with intervals ranging from 0.5s to 2.5s. Both the sender and receivers were never moved during the whole experiment. Without the sampling drift, the number of samples collected between two beeps on sender and receiver sides should be exactly the same (e.g., like the case of two Galaxy Nexus phones).

However, we indeed observed the drifts between phones of different brands. Table 5.1 shows the rounded mean values of ten experiments. The results are quite stable. Among these ten experiments, the maximum deviate we observed is only one sample. The positive values means the sampling rate of the sender is faster than that of the receiver and the negative values means the opposite. We also changed the roles of sender and receiver for each phone, the same drifts were observed.

From the table, we see the drifts linearly increase over time. This validates our compensation approach described in Section 5.3. That is why the last beep has to

Galaxy Nexus	0.5s	1.0s	1.5s	2.0s	2.5s
Galaxy Nexus	0	0	0	0	0
HTC EVO 3D	3	5	7	9	11
Nexus S	2	3	5	7	9
Galaxy S2	-4	-8	-12	-16	-20

Table 5.1: Drifts in terms of number of samples over time under 44.1kHz sampling rate

be emitted when phone is requested to move back to the origin.

5.7.3 Accuracy of determining direction

Line gesture. As mentioned in Section 5.4, drawing a line is the best for targets located in the center region. From Section 5.7.1, we also know that drawing a line could introduce less displacement measurement errors. Due to these reasons, we first tested our localization algorithms with a line gesture. The gesture radius is about 20cm, so two ending points of the line are located at (-20,0), (20,0). Consequently, four beeps are played at (0,0), (-20,0), (20,0) and back to (0,0) (all units are centimeters). The last beep is used to compensate the unknown sampling drifts. Two beep intervals are calculated based on the first-second beep pair and the first-third beep pair to derive corresponding delta distances. Receivers are placed along a line which is perpendicular to the moving line, each with 41cm away from the other. The tested orientations of receivers thereby range from -38° to 35° with respect to y axis in clockwise rotation. Each location is tested 5 times.

Fig. 5.11 shows the results. We found that the average angle errors were within 2.5° and the accuracy decreased when the receivers departed from the center. This is consistent with the the analysis in Section 5.4. It should be noticed that we indeed tested the cases where receivers were located out of the center region. The results are bad that we often could not find a solution, hence the results were not plotted in the figure. In summary, a line gesture achieves good performance for locating targets in the center region, but bad in the following cases. First, when the

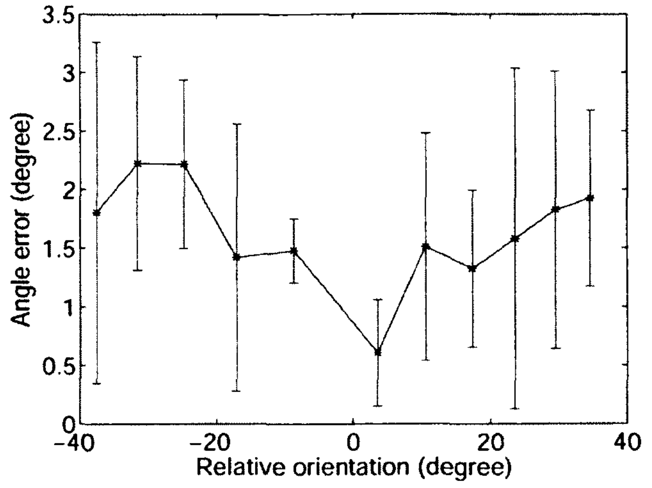


Figure 5.11: Angle errors with line gesture given targets are located in the left region

angle towards y axis is larger than 40 degree, the degrades significantly. Second, a line gesture may not distinguish between the front and back region.

Triangle gesture. To overcome the above problems, a triangle gesture with five beeps is used. A regular triangle only incurs three beeps at each corner. However, our moving strategy that the movement starts from the middle of the bottom side can produce five beeps, thus including the same advantages of line gestures. In following experiments, we investigated our scheme in several locations where a line gesture cannot work well. The moving radius is kept the same and the trial is from the coordinates (0,0) to (-20,0) to (0,20) to (20,0) then back to (0,0). The receivers is set to the coordinates (-264,-366), (-264,-244), (-264,-122), (-264,0), (-264,122), (-264,244) and (-264,-366) respectively. Each location is tested 5 times.

The results are shown in Fig. 5.12. It is seen that the average angle errors are less than 6 degree. Owing to the symmetry, similar results are obtained for receivers located in the right panel. Considering both Fig. 5.11 and Fig. 5.12, our localization algorithm can achieve less than 6° angle error in general given gesture radius equal to 20cm.

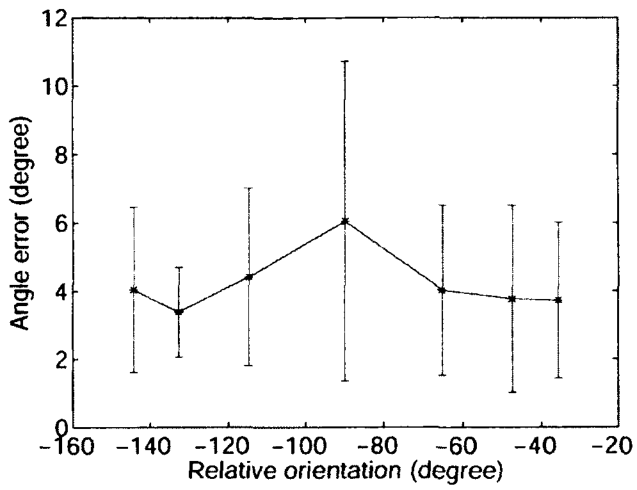


Figure 5.12: Angle errors with triangle gesture given targets are located in the left region

Moving radius. In addition to the shape of gestures, the moving radius also has effect on the accuracy. As a rule of thumb, large radius will incur less localization errors. We validate this rule in the following experiments, where receivers are placed in the center of a sender. We tested triangles with different moving radius. First, the movement is from the coordinates (0,0) to (-20,0) to (0,20) to (20,0) and back to (0,0). Next, the moving radius increases, and the moving trail is from (0,0) to (-40,0) to (0,40) to (40,0) and back to (0,0). Fig. 5.13 depicts the CDF of angle errors. We see that the gesture with radius equal to 40cm achieves better performance. Note that the mean values here are worse than the results shown in Fig. 5.11, because the outputs in our algorithm for more than three beeps are averaged by any possible solutions. Accurate results are deteriorated by the relatively inaccurate results.

5.7.4 Accuracy of determining distance

Our localization algorithms can estimate the direction of receivers accurately, but in poor estimation of the distance. That is because the relative movement of sender's phone is too small compared to the distances between sender and receivers. A

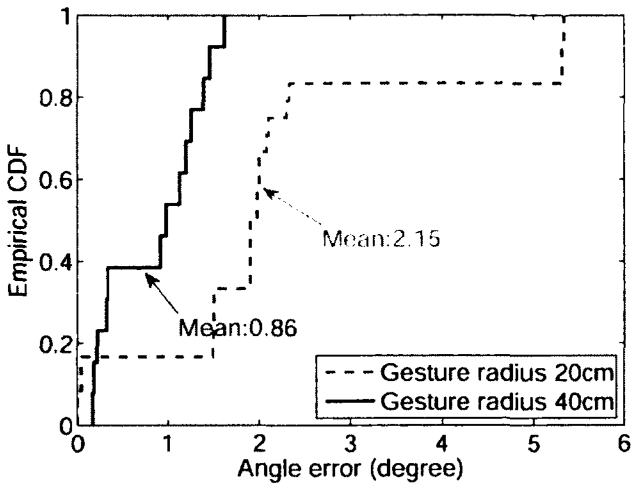


Figure 5.13: Angle errors against different moving radius

slight measurement error either in IMU sensors or acoustic systems may cause a significant departure from the exact point. The larger the distances between sender and receivers, the bigger the errors. According to our experience, the average errors are around 1m when receivers are 2m far away from the sender, and 3m when receivers are 5m far.

However, to locate group users, we believe that the order of distance is more useful than the absolute distance. In the case of multiple users located in the same direction, our localization scheme is still worthy as long as the order of distance to the sender is correctly determined. In this section, we present the experiment results that shows how far two users are apart away in the same direction, our algorithm can detect the correct order with more than 90% probability. This metric is also known as the localization resolution.

From Fig. 5.14, we found that given the moving radius equal to 20cm, the localization resolution is about 30cm within 1m distance between a sender and a receiver. That means given a receiver is 1m far away from the sender, when the distance of another user to the sender is within 70cm our algorithm can determine who is closer with high probability. Otherwise if two receivers are within 30cm

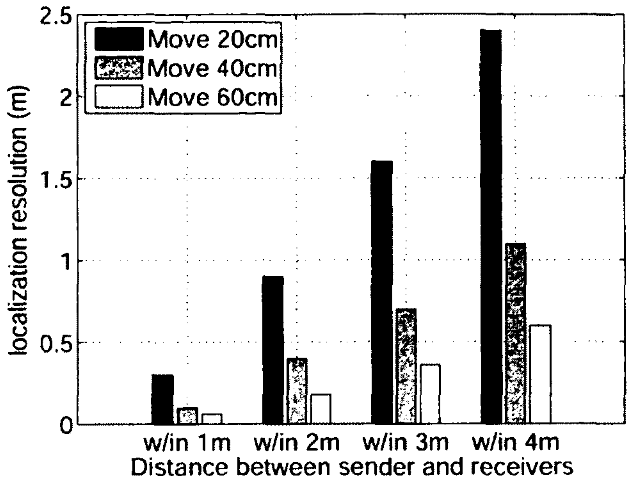


Figure 5.14: Localization resolution versus relative distance between sender and receivers under variant moving radius

distance, our algorithm will fail to determine the order. The localization resolution degrades when the relative distance increases. However, the larger moving radius is required if high resolution is needed.

5.7.5 Computation time

The finish time of localization process consists of two parts: the movement duration and computation time. When a phone is moving, samples collected from the microphone and IMU sensors are stored in memory or disk (when memory is full). The entire duration varies according to the shape and radius of the gesture, but it typically can be finished within 10s. Our algorithm then calculates the moving trail based on sensor readings, beep intervals based on microphone readings, and eventually the coordinates of the receivers. Given that all samples are in memory, the cross-correlation method is first tested then followed by our method. Table 5.2 lists the results. It is seen that our FFT-based correlation method can reduce the finishing time by more than 90%.

Gesture		Line(4 beeps)	Triangle (5 beeps)
Samples	Mean	389734	427008
	Std	16249	9523
Finish time (s)	Cross-correlation (Mean)	46.06	50.4
	Cross-correlation (std)	1.78	1.16
	SG-LOC (mean)	3.09	3.68
	SG-LOC (std)	0.07	0.05

Table 5.2: Comparison of finish time of detecting different number of beeps: traditional cross-correlation method versus our SG-LOC method

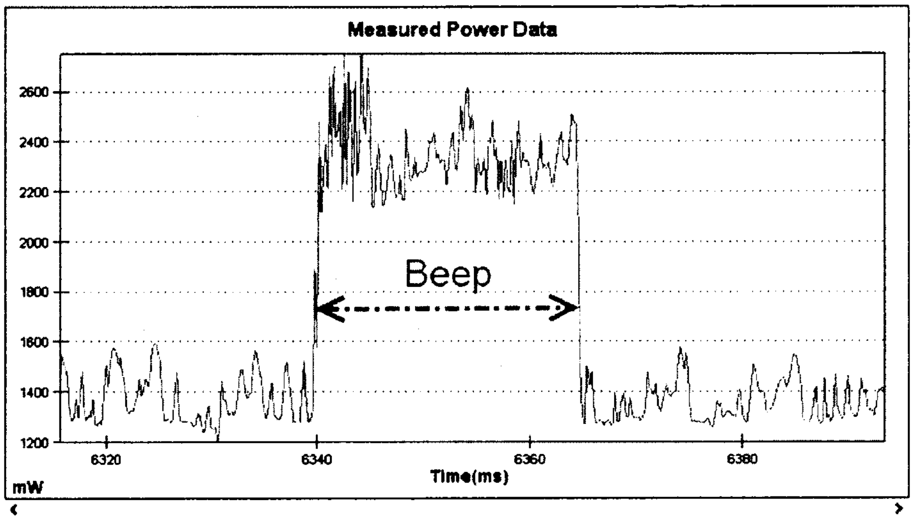


Figure 5.15: Power consumption of emitting a beep sound

5.7.6 Power consumption

In our system, a significant contributor to battery drain is the speaker when emitting beeps. Fig. 5.15 plots the consumed power for a beep. It is seen that the average power jumps from around 1300mW to 2300mW during 25ms (i.e., the duration of a beep). This overhead also motivates our work to reduce the number of beeps.

We measured the power consumption of our system on both the sender and receiver sides. For locating others, the sender has to turn on the speaker, the microphone and IMU sensors, but the receiver only needs to open the microphone.

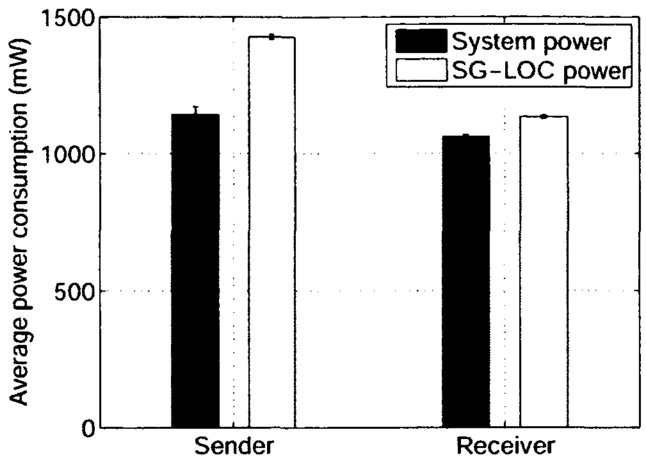


Figure 5.16: Comparison of different moving strategy

As a result, the sender is expected to consume more energy than the receiver. Fig. 5.16 shows the power consumption. In comparison the system power without SG-LOC is measured when both the screen and Wi-Fi are turned on. As we see, the power consumption for the sender increased by 24% (from 1143mW to 1429mW) when our system is running, while the power consumption for the receiver merely increased by 6% (from 1062mW to 1134mW).

Another significant source of power consumption is the computation overhead. Hence, we also measured the power for the computation separately. From the results, we found the computation usually incurred around 900mW power consumption.

5.7.7 Field test

Finally, we conducted a field test in our office room with 3mx5m size to measure the overall accuracy of our system. Five smartphones are placed on the table (seen in Fig. 5.17), a user is requested to draw a triangle gesture using the phone without any other limitation. The results show the estimated coordinates of four phones have less than 1° direction errors and less than 20cm distance errors on

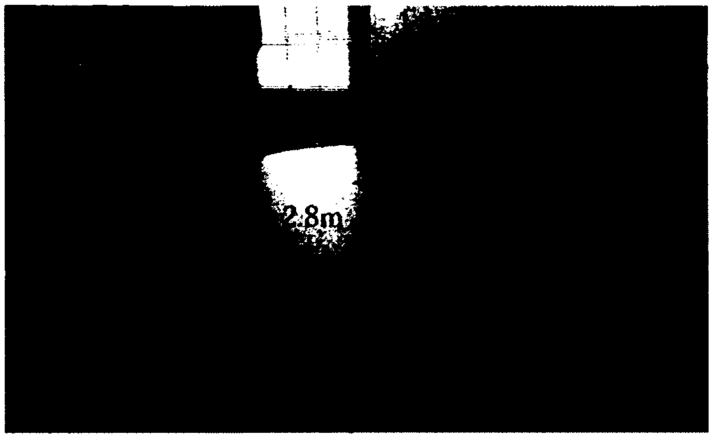


Figure 5.17: Field test environment

average. The distance errors are reasonably small so that the smartphones are placed correctly on the map. When the map is transmitted to all the users, each other users can identify each other correctly by viewing the locations.

5.8 Discussions

Continuously tracking. SG-LOC is originally designed for locating and grouping users in the network setup stage. To enable the continuous track of users after that, the localization scheme proposed in SG-LOC can be applied periodically. The alternative method to achieve continuously tracking is to leverage IMU sensors. Any rotation or movement of the phone will be detected by the sensors and updated to other users.

Accurate distance measurement. Our localization scheme can find accurate orientations of other users but achieve relatively less distance accuracy. To mitigate this problem, our scheme can be performed on multiple devices. With the angle information of each pair of devices, a more accurate distance can be derived.

Extension to 3D. In current implementation, 2D location is provided but our scheme is not limited to 2D only. With more beeps emitted during the movement, our positioning algorithm can be extended to 3D space with little modification. This will be studied in our future work.

5.9 Conclusion

In this chapter, we consider the problem of efficiently and securely grouping and locating mobile phone users in proximity. A system called SG-LOC is proposed to leverage a simple gesture to perform localization during network setup. By using internal motion sensors and speakers/microphones, our scheme combines gesture detection and acoustic techniques for a user to locate other users in an efficient, low-cost and scalable manner. We have designed, implemented and evaluated our system on commercial smartphones. The extensive experiments have shown that SG-LOC can achieve centimeter-level accuracy.

6 Conclusion and Future Work

Mobile computing has played an important role in the modern world. Wireless communication, allowing mobile devices to connect to networks and fetch remote resources, is an indispensable function to mobile computing. Massively popular mobile applications rely on wireless communication to perform their functions. The enhancements of existing wireless technologies will directly advance mobile technologies. Among all the existing problems, security, energy-efficiency, and location-awareness are the most important for the following reasons. First, many mobile devices access large amount of sensitive data under risky circumstances. This personal data is not protected by comprehensive security mechanisms because of the limited resources of mobile devices. Second, today's mobile devices increasingly perform many complex tasks, thus consuming more energy than ever before. Given the fact that battery technology has hardly been improved while wireless communication is a big contributor to the battery drain, it is urgent to reduce the power consumption while retaining network performance. Third, traditional location-oblivious wireless communication has become inadequate to meet the increasing demand of mobility. Location-awareness will lead to a more intuitive communication experience and improve the quality of communication. However, little research considers infrastructure-less localization. These problems have impeded the advance of mobile systems.

The goal of this dissertation is to mitigate the challenges of security, energy and localization for mobile computing. Through three communication patterns in 802.11 wireless networks, we investigate a collection of disparate-but-interrelated problems that are faced in today's wireless management. We enhance the security of the association mechanism for wireless clients to avoid connecting to malicious rogue APs under various network conditions. This is the first work that does not require administrator or infrastructure support. We also study the problem of power-saving for mobile devices working as software-defined APs. Since AP mode was recently introduced to mobile devices, there is little research in this domain. Our proposed technique will shed light upon the general research that explores comprehensive energy solutions for mobile computing systems. Lastly, we investigate the practical feasibility of location-aware wireless communication and design a fine-grained, efficient, lightweight and scalable localization algorithm. This technique allows mobile devices to know the relative location of others in a network without the support of network infrastructure and therefore has more advantages than existing solutions.

The following guidelines derived from our studies are intended to improve the design of advanced mobile computing systems.

- In a mobile computing environment, any device is potentially untrusted. Before establishing connectivity, a mobile device should verify the authenticity of the other side in its own right. This helps prevent unauthorized individuals from masquerading as legitimate devices and gaining access to sensitive data.
- Hardware technology behind mobile computing improves continuously. This constantly invalidates old wisdom, so we should challenge existing solutions to keep the system design consistent with the advance of hardware.
- When developing power saving systems, we should first consider the charac-

teristics of their applications. The application-specific design typically achieves more power savings than the general design.

- An important feature of today's mobile devices is the integration of diverse sensors on broad. This sensor data may provide the insight into user's context. Extracting the context to infer location can achieve very high accuracy and therefore is a promising research topic for mobile computing.

We have made some progress in improving the security, energy-efficiency and location-awareness of mobile computing. In the future, we will continue working on these challenging problems in wireless communication and mobile computing. For example, more and more mobile devices use cloud-based services to upload sensitive data and computational tasks for processing. How to overcome security concerns by redacting, removing or replacing sensitive data from records that are sent to the cloud becomes increasingly important. We attempt to continue research on this problem. Additionally, the improvement of hardware technologies will introduce complicated mobile devices (e.g., from single core to multi-cores). Energy efficiency will continue to be a source of interesting research problems. Finally, the increasing demand of mobility will drive the research on localization techniques to support various levels of location-awareness applications.

References

[1] Air defence. http://www.airdefence.net.

[2] Air magnet. http://www.airmagnet.com.

[3] Air wave. http://www.airwave.com.

[4] Android webkit package and webview class. http://developer.android.com/reference/android/webkit/package-summary.html.

[5] Cisco visual networking index (vni) global mobile data traffic forecast, 2012-2017. http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-520862.html.

[6] Deliberant cpe 2-12. http://www.streakwave.com/mmSWAVE1/Video/CPE202-12.pdf.

[7] Drive-thru internet. http://www.drive-thru-internet.org/.

[8] Google Mountain View. http://wifi.google.com.

[9] Ipw3945. http://ipw3945.sourceforge.net.

[10] Libpcap. http://www.tcpdump.org/.

[11] Madwifi. http://madwifi.org.

[12] Monsoon power monitor. http://www.msoon.com/LabEquipment/PowerMonitor/.

[13] Mywi and mywi ondemand. `http://intelliborn.com/mywi.html`.

[14] Network driver interface specification (NDIS). `http://msdn.microsoft.com/en-us/library/ff559102.aspx`.

[15] Openwrt. `https://openwrt.org/`.

[16] Wireless lan medium access control (MAC) and physical layer (PHY) specifications. IEEE Std 802.11, 2007.

[17] Atul Adya, Paramvir Bahl, Ranveer Chandra, and Lili Qiu. Architecture and techniques for diagnosing faults in IEEE 802.11 infrastructure networks. In *Mobicom 2004*.

[18] Sandip Agrawal, Ionut Constandache, Shravan Gaonkar, Romit Roy Choudhury, Kevin Caves, and Frank DeRuyter. Using mobile phones to write in air. In *MobiSys*, pages 15--28, 2011.

[19] M. Anand, E. Nightingale, and J. Flinn. Self-tuning wireless network power management. In *Mobicom*, 2003.

[20] Paramvir Bahl, Ranveer Chandra, Jitendra Padhye, Lenin Ravindranath, Manpreet Singh, Alec Wolman, and Brian Zill. Enhancing the security of corporate Wi-Fi networks using DAIR. In *MobiSys 2006*.

[21] Paramvir Bahl and Venkata N. Padmanabhan. Radar: An in-building rf-based user location and tracking system. In *INFOCOM*, pages 775--784, 2000.

[22] Nilanjan Banerjee, Sharad Agarwal, Paramvir Bahl, Ranveer Chandra, Alec Wolman, and Mark Corner. Virtual compass: relative positioning to sense mobile social interactions. In *Pervasive*, pages 1--21, 2010.

[23] D. Bertozzi, L. Benini, and B. Ricco. Power aware network interface management for streaming multimedia. In *WCNC*, 2002.

[24] Raheem Beyah, Shantanu Kangude, George Yu, Brian Strickland, and John Copeland. Rogue access point detection using temporal traffic characteristics. In *Globecom 2004*.

[25] Gaetano Borriello, Alan L. Liu, Tony Offer, Christopher Palistrant, and Richard Sharp. Walrus: wireless acoustic location with room-level resolution using ultrasound. In *MobiSys*, pages 191--203, 2005.

[26] Vladimir Brik, Suman Banerjee, Marco Gruteser, and Sangho Oh. Wireless device identification with radiometric signatures. In *Mobicom 2008.*

[27] Broadcom. BCM4329 product brief. `http://www.broadcom.com`.

[28] Vladimir Bychkovsky, Bret Hull, Allen K. Miu, Hari Balakrishnan, and Samuel Madden. A Measurement Study of Vehicular Internet Access Using In Situ Wi-Fi Networks. In *12th ACM MOBICOM Conf.*, Los Angeles, CA, September 2006.

[29] Daniel Camps-Mur, Xavier Pérez-Costa, and Sebastií Sallent-Ribes. Designing energy efficient access points with wi-fi direct. *Comput. Netw.*, 55(13):2838--2855, 2011.

[30] Yin Chen, Dimitrios Lymberopoulos, Jie Liu, and Bodhi Priyantha. Fm-based indoor localization. In *MobiSys*, pages 169--182, 2012.

[31] Krishna Chintalapudi, Anand Padmanabha Iyer, and Venkata N. Padmanabhan. Indoor localization without the pain. In *Proceedings of the sixteenth annual international conference on Mobile computing and networking*, MobiCom '10, pages 173--184, New York, NY, USA, 2010. ACM.

[32] Krishna Chintalapudi, Anand Padmanabha Iyer, and Venkata N. Padmanabhan. Indoor localization without the pain. In *MobiCom*, pages 173--184, 2010.

[33] F. Dogar, P. Steenkiste, and K. Papagiannaki. Catnap: Exploit high bandwidth wireless interfaces to save energy for mobile devices. In *Mobisys*, 2010.

[34] Jonathan Friedman, Zainul Charbiwala, Thomas Schmid, Young Cho, and Mani Srivastava. M.: Angle-of-arrival assisted radio interferometry (ari) target localization, 2008.

[35] Sumit Garg, Manish Kalia, and Rajeev Shorey. Mac scheduling policies for power optimization in bluetooth: A master driven tdd wireless system. In *IEEE VTC*, 2000.

[36] Anastasios Giannoulis, Marco Fiore, and Edward W. Knightly. Supporting vehicular mobility in urban multi-hop wireless networks. In *MobiSys*, 2008.

[37] Lewis Girod, Martin Lukac, Vlad Trifa, and Deborah Estrin. The design and implementation of a self-calibrating distributed acoustic sensing platform. In *SenSys*, pages 71--84, 2006.

[38] Abhishek Goswami, Luis E. Ortiz, and Samir R. Das. Wigem: a learning-based approach for indoor localization. In *CoNEXT*, pages 3:1--3:12, 2011.

[39] Hao Han, Bo Sheng, Chiu Tan, Qun Li, and Sanglu Lu. A measurement based rogue AP detection scheme. In *The 28th IEEE International Conference on Computer Communications*, Rio de Janeiro, Brazil, 2009.

[40] Hao Han, Bo Sheng, Chiu C. Tan, Qun Li, and Sanglu Lu. A measurement based rogue ap detection scheme. In *Infocom 2009*.

[41] Hao Han, Fengyuan Xu, Chiu Chiang Tan, Yifan Zhang, and Qun Li. Defending against vehicular rogue aps. In *INFOCOM*, pages 1665--1673, 2011.

[42] Andy Harter, Andy Hopper, Pete Steggles, Andy Ward, and Paul Webster. The anatomy of a context-aware application. In *MobiCom*, pages 59--68, 1999.

[43] Mike Hazas and Andy Ward. A novel broadband ultrasonic location system. In *Ubicomp*, pages 264--280, 2002.

[44] Yong He and Ruixi Yuan. A novel scheduled power saving mechanism for 802.11 wireless lans. *IEEE Trans. Mob. Comput.*, 8(10):1368--1383, 2009.

[45] Junxian Huang, Qiang Xu, Birjodh Tiwana, Z. Morley Mao, Ming Zhang, and Paramvir Bahl. Anatomizing application performance differences on smartphones. In *MobiSys*, 2010.

[46] Suman Jana and Sneha Kasera. On fast and accurate detection of unauthorized wireless access points using clock skews. In *Mobicom 2008.*

[47] Amit P. Jardosh, Krishna N. Ramachandran, Kevin C. Almeroth, and Elizabeth M. Belding-Royer. Understanding congestion in IEEE 802.11b wireless networks. In *Internet Measurment Conference*, pages 279--292, 2005.

[48] Jaeyeon Jung, Emil Sit, Hari Balakrishnan, and Robert Morris. DNS performance and the effectiveness of caching. *Computer Communication Review*, 32(1):74, 2002.

[49] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The click modular router. In *TOCS 2000.*

[50] R. Krashinsky and H. Balakrishnan. Minimizing energy for wireless web access with bounded slowdown. In *Mobicom*, 2002.

[51] Branislav Kusy, János Sallai, György Balogh, Ákos Lédeczi, Vladimir A. Protopopescu, Johnny Tolliver, Frank DeNap, and Morey Parang. Radio interferometric tracking of mobile wireless nodes. In *MobiSys*, pages 139--151, 2007.

[52] Patrick Lazik and Anthony Rowe. Indoor pseudo-ranging of mobile devices using ultrasonic chirps. In *SenSys*, pages 99--112, 2012.

[53] R. LiKamWa, Y. Liu, N. D. Lane, and L. Zhong. Can your smartphone infer your mood? In *PhoneSense workshop*, 2011.

[54] Ting-Yu Lin and Yu-Chee Tseng. An adaptive sniff scheduling scheme for power saving in bluetooth. *IEEE Wireless Communications*, 9(6):92--103, 2002.

[55] J. Liu and L. Zhong. Micro power management of active 802.11 interfaces. In *Mobisys*, 2008.

[56] Hong Lu, Jun Yang, Zhigang Liu, Nicholas D. Lane, Tanzeem Choudhury, and Andrew T. Campbell. The jigsaw continuous sensing engine for mobile phone applications. In *SenSys*, pages 71--84, 2010.

[57] Liran Ma, Amin Y. Teymorian, and Xiuzhen Cheng. A hybrid rogue access point protection framework for commodity Wi-Fi networks. In *Infocom 2008.*

[58] Chad D. Mano, Andrew Blaich, Qi Liao, Yingxin Jiang, David A. Cieslak, David Salyers, and Aaron Striegel. Ripps: Rogue identifying packet payload slicer detecting unauthorized wireless hosts through network traffic conditioning. *ACM Trans. Inf. Syst. Secur.*, 11(2), 2008.

[59] J. Manweiler and R. R. Choudhury. Avoiding the rush hours: Wifi energy mangement via traffic isolation. In *Mobisys*, 2011.

[60] MiFi. `http://en.wikipedia.org/wiki/MiFi`.

[61] Rajalakshmi Nandakumar, Krishna Kant Chintalapudi, and Venkata N. Padmanabhan. Centaur: locating devices in an office environment. In *Mobicom*, pages 281--292, 2012.

[62] Michael Neufeld, Jeff Fifield, Christian Doerr, and Anmol Sheth andDirk Grunwald. Softmac - flexible wireless research platform. In *HotNets-IV*, 2005.

[63] Lionel M. Ni, Yunhao Liu, Yiu Cho Lau, and Abhishek P. Patil. Landmarc: Indoor location sensing using active rfid. *Wireless Networks*, 10(6):701--710, 2004.

[64] Jörg Ott and Dirk Kutscher. Drive-thru internet: Ieee 802.11b for "automobile" users. In *INFOCOM*, 2004.

[65] Utpal Paul, Riccardo Crepaldi, Jeongkeun Lee, Sung-Ju Lee, and Raúl H. Etkin. Characterizing wifi link performance in open outdoor networks. In *SECON*, pages 251--259, 2011.

[66] Chunyi Peng, Guobin Shen, Yongguang Zhang, Yanlin Li, and Kun Tan. Beepbeep: a high accuracy acoustic ranging system using cots mobile devices. In *SenSys*, pages 1--14, 2007.

[67] Chunyi Peng, Guobin Shen, Yongguang Zhang, and Songwu Lu. Point&#38;connect: intention-based device pairing for mobile phone users. In *MobiSys*, pages 137--150, 2009.

[68] C. Poellabauer and K. Schwan. Energy-aware traffic shaping for wireless realtime applications. In *RTAS*, 2004.

[69] Nissanka B. Priyantha, Anit Chakraborty, and Hari Balakrishnan. The cricket location-support system. In *MobiCom*, pages 32--43, 2000.

[70] D. Qiao and K. Shin. Smart power-saving mode for ieee 802.11 wireless lans. In *Infocom*, 2005.

[71] Jian Qiu, David Chu, Xiangying Meng, and Thomas Moscibroda. On the feasibility of real-time phone-to-phone 3d localization. In *SenSys*, pages 190--203, 2011.

[72] E. Rozner, V. Navda, R. Ramjee, and S. Rayanchu. Napman: Network-assisted power management for wifi devices. In *Mobisys*, 2010.

[73] Souvik Sen, Bozidar Radunovic, Romit Roy Choudhury, and Tom Minka. You are facing the mona lisa: spot localization using phy layer information. In *MobiSys*, pages 183--196, 2012.

[74] Ashish Sharma, Vishnu Navda, Ramachandran Ramjee, Venkata N. Padmanabhan, and Elizabeth M. Belding. Cool-tether: energy efficient on-the-fly wifi hot-spots using mobile phones. In *CoNEXT*, pages 109--120, 2009.

[75] Yong Sheng, Keren Tan, Guanling Chen, David Kotz, and Andrew Campbell. Detecting 802.11 MAC layer spoofing using received signal strength. In *Infocom 2008*.

[76] Sachin Shetty, Min Song, and Liran Ma. Rogue access point detection by analyzing network traffic characteristics. In *Milcom 2007*.

[77] Stephen P. Tarzia, Peter A. Dinda, Robert P. Dick, and Gokhan Memik. Indoor localization without infrastructure using the acoustic background spectrum. In *MobiSys*, pages 155--168, 2011.

[78] Tcpdump. `http://www.tcpdump.org/`.

[79] Agilent technologies. 89600s series VXI-based vector signal analyzer.

[80] Tijs van Dam and Koen Langendoen. An adaptive energy-efficient mac protocol for wireless sensor networks. In *SenSys*, pages 171--180, 2003.

[81] Dimitris Vassis, George Kormentzas, Angelos N. Rouskas, and Ilias Maglogiannis. The ieee 802.11g standard fo high data rate wlans. *IEEE Network*, 19(3):21--26, 2005.

[82] Dimitris Vassis, George Kormentzas, Angelos N. Rouskas, and Ilias Maglogiannis. The ieee 802.11g standard fo high data rate wlans. *IEEE Network*, 19(3):21--26, 2005.

[83] Aravind Venkataraman and Raeem Beyah. Rogue access point detection using innate characteristics of the 802.11 mac. In *SecureComm 2009.*

[84] Lanier Watkins, Raheem Beyah, and Cherita Corbett. A passive approach to rogue access point detection. In *Globecom 2007.*

[85] Wei Wei, Sharad Jaiswal, James F. Kurose, and Donald F. Towsley. Identifying 802.11 traffic from passive measurements using iterative bayesian inference. In *INFOCOM*, 2006.

[86] Wei Wei, Kyoungwon Suh, Bing Wang, Yu Gu, Jim Kurose, and Don Towsley. Passive online rogue access point detection using sequential hypothesis testing with TCP ACK-pairs. In *IMC 2007.*

[87] H. Wirtz, R. Backhaus, R. Hummen, and K. Wehrle. Establishing mobile adhoc networks in 802.11 infrastructure mode. In *WiNTECH demo session*, 2011.

[88] Starsky H. Y. Wong, Hao Yang, Songwu Lu, and Vaduvur Bharghavan. Robust rate adaptation for 802.11 wireless networks. In *Mobicom 2006.*

[89] Yan Wu, Sonia Fahmy, and Ness B. Shroff. Optimal sleep/wake scheduling for time-synchronized sensor networks with qos guarantees. *IEEE/ACM Trans. Netw.*, 17(5):1508--1521, 2009.

[90] T. D. Todd Y. Li and D. Zhao. Access point power saving in solar/battery powered ieee 802.11 ess mesh networks. In *IEEE QShine*, 2005.

[91] Jie Yang, Simon Sidhom, Gayathri Chandrasekaran, Tam Vu, Hongbo Liu, Nicolae Cecan, Yingying Chen, Marco Gruteser, and Richard P. Martin. Detecting driver phone use leveraging car speakers. In *MobiCom*, pages 97--108, 2011.

[92] Zheng Yang, Chenshu Wu, and Yunhao Liu. Locating in fingerprint space: wireless indoor localization with little human intervention. In *MOBICOM*, pages 269--280, 2012.

[93] Hongda Yin, Guanling Chen, and Jie Wang. Detecting protected layer-3 rogue APs. In *Broadnets 2007.*

[94] Moustafa Youssef and Ashok Agrawala. The horus wlan location determination system. In *MobiSys*, pages 205--218, 2005.

[95] Moustafa Youssef and Udaya Shankar. Pinpoint: An asynchronous time-based location determination system. In *MobiSys*, pages 165--176. ACM Press, 2006.

[96] Feng Zhang, Terence D. Todd, Dongmei Zhao, and Vytas Kezys. Power saving access points for ieee 802.11 wireless network infrastructure. *IEEE Trans. Mob. Comput.*, 5(2):144--156, 2006.

[97] Zengbin Zhang, David Chu, Xiaomeng Chen, and Thomas Moscibroda. Swordfight: enabling a new class of phone-to-phone action games on commodity phones. In *MobiSys*, pages 1--14, 2012.

VITA

Hao Han received his Bachelor of Science degree in Computer Science from Nanjing University, China, in 2005. He was admitted to the Ph.D. program in Computer Science Department at the College of William and Mary in 2009, and became a Ph.D. candidate in 2010. His major research interests span many areas of wireless networks, mobile computing and RFID systems. He is particularly interested in the design and implementation of techniques to improve the security, power-efficiency and cognition of mobile systems.