2015

Providing efficient services for smartphone applications

Yifan Zhang
College of William & Mary - Arts & Sciences

Providing Efficient Services for Smartphone Applications

Yifan Zhang

Liuzhou, Guangxi, China

Bachelor of Engineering, Beihang University, 2004

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
August, 2014

APPROVAL PAGE

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

Doctor of Philosophy

Yifan Zhang

Approved by the Committee, August, 2014

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

Associate Professor Peter Kemper, Computer Science
The College of William and Mary

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

Associate Professor Haining Wang, Computer Science
The College of William and Mary

Associate Professor Gexin Yu, Mathematics
The College of William and Mary

ABSTRACT

Mobile applications are becoming an indispensable part of people's lives, as they allow access to a broad range of services when users are on the go. We present our efforts towards enabling efficient mobile applications in smartphones. Our goal is to improve efficiency of the underlying services, which provide essential functionality to smartphone applications. In particular, we are interested in three fundamental services in smartphones: wireless communication service, power management service, and location reporting service.

For the wireless communication service, we focus on improving spectrum utilization efficiency for cognitive radio communications. We propose ETCH, a set of channel hopping based MAC layer protocols for communication rendezvous in cognitive radio communications. ETCH can fully utilize spectrum diversity in communication rendezvous by allowing all the rendezvous channels to be utilized at the same time.

For the power management service, we improve its efficiency from three different angles. The first angle is to reduce energy consumption of WiFi communications. We propose HoWiES, a system for WiFi energy saving by utilizing low-power ZigBee radio. The second angle is to reduce energy consumption of web based smartphone applications. We propose CacheKeeper, which is a system-wide web caching service to eliminate unnecessary energy consumption caused by imperfect web caching in many smartphone applications. The third angle is from the perspective of smartphone CPUs. We found that existing CPU power models are ill-suited for modern multicore smartphone CPUs. We present a new approach of CPU power modeling for smartphones. This approach takes CPU idle power states into consideration, and can significantly improve power estimation accuracy and stability for multicore smartphones.

For the location reporting service, we aim to design an efficient location proof solution for mobile location based applications. We propose VProof, a lightweight and privacy-preserving location proof scheme that allows users to construct location proofs by simply extracting unforgeable information from the received packets.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

First of all, my sincere appreciation goes to my advisor and dissertation committee chair, Dr. Qun Li. Qun led me into the wonderful academic world, and has guided and inspired me every step along the way towards this dissertation. He has always been there offering me his insightful advice and encouragement when I needed them most. The most precious experience I learned from him is being persistent in working hard towards our goal, which would eventually help us sail to the destination. This six yeas' experience of working with Qun is invaluable to me, and will continue to influence my future career.

I would like to thank my committee members, Dr. Peter Kemper, Dr. Weizhen Mao, Dr. Haining Wang and Dr. Gexin Yu, for their effort and advice in helping me improve this dissertation. I would like to thank Dr. Gexin Yu, for his tremendous help in my first research project, which set a good start of my Ph.D study. I would like to deliver my special thanks to committee members, Dr. Peter Kemper, Dr. Weizhen Mao and Dr. Haining Wang, as well as my internship advisor at Microsoft Research Dr. Yunxin Liu, for their assistance in my job search process.

I am grateful to have the experience of working with many brilliant people here at William and Mary. I would like to thank the three senior members of the research group, Haodong Wang, Bo Sheng, and Chiu Tan, for their help when I first entered the program. My special thanks goes to Chiu, for the many discussions and suggestions that helped me grow as a researcher. I would like to thank Fengyuan Xu, Wei Wei, Hao Han, Zhengrui Qin for their support and friendship. Being able to having them around for almost my entire Ph.D had made my work and life much easy and enjoyable. I would also like to thank all the past and current members of my research group, including Lei Xie, Xiaojun Zhu, Baosheng Wang, Huda El Hag Mustafa, Ed Novak, Nancy Carter, Zijiang Hao, Yutao Tang, Shanhe Yi and Cheng Li. It is always been my pleasure to have the opportunity to work with them.

I appreciate the hard work of the staff in the Computer Science Department at William and Mary. In particular, I would like to thank Vanessa Godwin, Jacqulyn Johnson and Dale Hayes, for all their help during my Ph.D. I would like to thank Dr. Phil Kearns and the Techie team, for their hard work of maintaining a smooth and state-of-art work environment for us.

Finally, I would like to express my deepest gratefulness to my dear family. It is undoubted that this dissertation would not have been possible without their firm support over the years. I would like to especially thank my wife, Yu Peng, who has been always at my side and helped me go through each and every tough moment during my PhD. I would like to thank the four parents, mine and Yu's, for their unconditional love. This work is dedicated to my beloved family.

To my beloved family, especially to my parents, Bingkun Zhang and Aimin Li.

LIST OF TABLES

LIST OF FIGURES

1 Introduction

1.1 Overview

Since the emergence of smartphones, users have been able to enjoy traditional mobile applications (e.g., phone calls and SMS) as well as many new applications with their mobile phones. Smartphone applications are now tapping into virtually every aspect of people's lives, spanning from online banking to health monitoring, from news reading to mobile gaming, and from trip planning to video streaming. Applications that can efficiently utilize smartphone resources (e.g., CPU, battery, and network bandwidth) are important to both smartphone owners and service providers. Because they can not only bring better user experience (e.g., smaller response time and longer battery life) for smartphone owners, but also lead to more optimal resource usage (e.g., better bandwidth utilization ratio and smaller computation overhead) for mobile service providers. Therefore, enabling efficient mobile applications in smartphones has been an important topic in mobile computing research.

Smartphone applications rely on different services provided by the smartphone operating system to offer good user experience and achieve optimal resource usage. Each service provides a specific kind of functionality that is necessary for many applications running in the phone. Wireless communication, battery power management, and location dependent applications are the three most important aspects that set smartphone based computing different from traditional PC com-

puting. Therefore, we are interested in improving the following smartphone services that are related to these three aspects.

- *Wireless communication service* provides wireless connectivity for smartphone applications. Compared to applications in PCs, the most salient feature of smartphone applications is that they allow a user to access resources in the Internet in a wireless manner or to connect to other users even when she is on the go. Most smartphone applications need to use at least one kind of wireless connection (e.g., cellular, WiFi and Bluetooth). A recent survey on 856 free Android applications and 100 paid Android applications reveals that 87% of free and 66% of paid Android applications request permissions of accessing wireless network [1]. Therefore, improving efficiency of wireless communication service is key to enhancing efficiency of smartphone applications.

- *Power management service* allows smartphone applications to use battery power efficiently. Because smartphones are powered by batteries, the usability of smartphone applications is affected by one important factor, which is not so significant in traditional PC computing - power efficiency. Therefore, improving efficiency of power management service in smartphones is important to providing good user experience.

- *Location reporting service* enables location based service (LBS) applications in smartphones. LBS is an important category of services enabled by the proliferation of smartphones. LBS utilizes smartphones' on-board chips/sensors, such as GPS, cellular, gyroscope and accelerometer, to report phones' geolocations, based on which mobile applications can provide customized services to the users. Examples of well-known LBS applications on the Android platform include `Google Maps` [2] (which provides map and real-time navigation services based on user's geolocations), `Facebook`

[3] and Foursquared [4] (both of which allow users to "check in" different venues and post comments and ratings about the venues), and GasBuddy [5] (through which users can report fuel prices they see and find the gas stations providing satisfying prices based on other users' reports). Improving efficiency of location report service is of great benefit to these LBS mobile applications.

In this dissertation, we present our efforts in improving efficiency of the above three most important services that differentiate smartphone based computing from traditional PC based computing. In the next section, we give an overview of the problem of each effort, as well as our contribution of solving the problem.

1.2 Problems and contributions

1.2.1 Achieving efficient rendezvous for cognitive radio communications

Wireless communication service is critical to the proper functioning of mobile applications. However, as smartphones and other mobile devices are getting popular, we are facing a severe wireless spectrum deficiency problem, where the unlicensed band are becoming too crowded to achieve good wireless communication performance. Cognitive radio is a promising technology to solve this spectrum scarcity problem. In this technology, cognitive radio transceivers can automatically detects available wireless channels, and changes its transmission or reception parameters accordingly so that more wireless communications may run concurrently in a given spectrum band [6]. With the introduction of cognitive radios and dynamic spectrum access (DSA for short) management, those previously closed licensed spectrum, whose bandwidth is much larger than that of unlicensed spectrum, can be opened to unlicensed wireless users without affecting the nor-

mal operations of the licensed users. Because of the great benefits that could be brought by cognitive radios, we believe that smartphones will be shipped with cognitive radios as the technology matures in the near future. Therefore, studying how to improve efficiency of cognitive radio communications is an important direction of providing good wireless communication service to mobile applications in smartphones.

In cognitive radio communications, unlicensed users (i.e., secondary users) are granted the access to the licensed spectrum if it is not being used by the licensed users (i.e., primary users). In the meantime, secondary users should yield the working wireless channel if the primary user appears. The dynamic availability of wireless spectrum requires secondary users to hop on different wireless channels, rather than staying in the same channel. Therefore, for two secondary users without knowing each other's working channel for communication, they need to first establish a *control channel*, upon which they exchange certain control information, such as communication channel and data rate, before they can start the data communication. The process of establishing control channel for the pair of secondary users is called *communication rendezvous*.

Existing solutions fall short of providing a solution for communication rendezvous that can efficiently utilize wireless spectrum. For example, the state-of-the-art channel hopping based solution can only guarantee one channel to be utilized at the same time [7]. To solve the problem, we propose *ETCH*, efficient channel hopping based MAC-layer protocols for communication rendezvous in cognitive radio communications [8, 9]. Compared to the existing solutions, ETCH fully utilizes spectrum diversity in communication rendezvous by allowing all the rendezvous channels to be utilized at the same time. We propose two protocols, SYNC-ETCH, which is a synchronous protocol assuming secondary users can synchronize their channel hopping processes, and ASYNC-ETCH, which is an asynchronous protocol not relying on global clock synchronization. Our theoreti-

cal analysis and `ns-2` based evaluation show that ETCH achieves better performances of time-to-rendezvous and throughput than the existing work.

1.2.2 Gaining energy savings for WiFi communications

Wireless communication service enables connectivity between smartphones. However, it is also a major source of power consumption in smartphones. An efficient power management service is important for prolonging battery life for smartphones. In this effort, we aim to improve smartphone power management service by gaining energy savings for WiFi communications.

WiFi is the one of the two most commonly used means for data transmission in smartphones (with the other one being cellular based data transmission). WiFi interface consumes a considerable amount of power when it is active, and is a major source of energy consumption affecting user experience. Therefore, studying how to gain energy savings from WiFi communications is a crucial step towards providing good power management service to mobile applications in smartphones.

We observe that there are three scenarios where a WiFi radio has to stay active without performing any real communications.

- *First*, a WiFi radio has to stay active to scan for networks in the scanning state. The power consumption for network scanning is considerably salient for the lack of WiFi coverage in many places.

- *Second*, during PSM (Power Save Mode) standby, a WiFi radio needs to constantly switch to active to receive wireless access point (AP) beacons and check if the AP has buffered its packets. Recent work [10, 11] show that users usually leave their smartphones idle for most of the time. The long idle time contributes to a non-negligible amount of WiFi energy consumption.

- *Third*, when waken up from PSM standby, a WiFi radio has to stay active doing nothing while waiting for its turn to communicate with the AP if there

are multiple devices contending for the channel.

The WiFi radio power consumptions in the above scenarios are significant: our measurements show that the power consumptions of WiFi scanning and PSM standby in a Samsung Galaxy S2 smartphone account for 65% and 11% of the entire system power consumption respectively, and recent work [12, 13] show that the wakeup contentions could cause up to four times more power consumption.

We propose *HoWiES*, a system that saves energy consumed by WiFi interfaces in mobile devices with the assistance of ZigBee radios [14, 15]. The key idea of HoWiES is that the operations of a WiFi radio in above scenarios can be delegated to a low power ZigBee radio. In this case, WiFi radio will be turned off when there is no packet to transmit and receive, and the ZigBee radio is responsible for discovering the presence of WiFi networks and detecting if the AP intends for the device to communicate. This way, the significant power consumptions on WiFi radio in those scenarios are reduced to the reasonably low power consumptions on ZigBee radio. The core component of HoWiES is a WiFi-ZigBee message delivery scheme that enables WiFi radios to convey different messages to ZigBee radios in mobile devices. Based on the WiFi-ZigBee message delivery scheme, we design three protocols that target at three WiFi energy saving opportunities in scanning, standby and wakeup respectively. We have implemented the HoWiES system with two mobile devices platforms and two AP platforms. Our real-world experimental evaluation shows that our system can convey thousands of different messages from WiFi radios to ZigBee radios with an accuracy over 98%, and our energy saving protocols, while maintaining the comparable wakeup delay to that of the standard 802.11 power save mode, save 88% and 85% of energy consumed in scanning state and standby state respectively.

1.2.3 Reducing energy consumption for web applications.

In this effort, we are trying to improve the power management service by reducing energy consumption caused at the application layer.

Web traffic is the dominant type of Internet traffic [16], and with the popularity of smartphones and tablets, an increasing amount of web traffic originates from mobile devices. The mobile web traffic has grown 35% in under a year [17], and now accounts for 20% of the U.S. web traffic [18]. Unlike conventional PCs, where web browser is the dominant source of web traffic, smartphones have another significant source of web traffic: dedicated mobile apps. The dedicated apps are getting popular because they provide users with convenient user interfaces that are tailored according to specific tasks and smartphones' physical constraints. The popularity of dedicated mobile apps has led online content providers to develop their own dedicated app to interact with their web services. Consequently, the diversity of apps and their developers has resulted in certain apps not being fully compliant with Internet standards or guidelines.

Among those guidelines, guideline on web caching is an important one for energy consumption and performance of web based applications. This is because an appropriate web caching implementation in mobile apps will benefit both users and network operators. With such an implementation, users can (a) conserve energy by reducing unnecessary data transmissions, (b) experience a higher quality of service, since the data can be accessed faster locally, and (c) lower costs, since users may have to pay a higher fee for downloading more data. Network operators also benefit when mobile apps implement web caching correctly since this reduces the congestion on the network, especially the last mile radio connections.

However, despite the importance of web caching, large numbers of mobile apps have *imperfect web caching*, meaning that web caching is either implemented for only certain HTTP resources the apps request, or is not implemented at all. The reason is that since apps without caching or with poor caching will still have the "look-and-feel", some developers will spend less time implementing and

testing the caching behavior of their apps.

We propose *CacheKeeper*, an OS web caching service transparent to mobile apps for smartphones [19]. CacheKeeper provides the correct web caching implementation with no effort on the part of mobile app developers. Developers do not need to install any additional libraries or incorporate any additional API calls to take advantage of CacheKeeper. Furthermore, CacheKeeper is backward compatible, meaning that existing apps can take advantage of CacheKeeper without any modifications. We implemented a prototype of CacheKeeper in Linux kernel, and evaluated it with extensive experiments. Our evaluation on 10 top ranked Android apps shows that our CacheKeeper prototype can save 42% HTTP traffic with real user browsing behaviors and reduce web accessing latency by half under real 3G settings.

1.2.4 Improving CPU power modeling for multicore smartphones.

CPU is a major source of power consumption in smartphones [20]. As multicore smartphones become increasingly popular, CPU power consumption becomes a much more significant component in the smartphone power consumption portfolio. For example, on a quad-core Samsung Galaxy S3 smartphone, the CPU power is as high as 2,845 mW, which is 2.53 times of the maximum power of the screen, and is 2.5 times of the maximum power of the 3G interface [21]. According to our measurements, the CPU power consumption of the Google Nexus series smartphones has increased significantly in the last three generations: the CPU power consumption of a Google Nexus 4 smartphone (quad-core, the 4th Nexus generation) could reach 4,065 mW, which is 2.03 times of the maximum CPU power of a Galaxy Nexus smartphone (dual-core, the 3rd Nexus generation), and is 4.51 times of that of a Nexus S smartphone (single-core, the 2nd Nexus generation). Therefore, accurate estimation and efficient management of CPU power consumption are among the most important issues in power management

of multicore smartphones.

Power modeling is a lightweight and effective approach to estimate power consumption of smartphone CPU. Proper and accurate power models of smartphone components benefit both users and developers. Accurate power models help to detect power hungry applications, and thus users get better battery life of their smartphones [22]. Accurate power models also help developers profile, and consequently optimize, the energy consumption of their smartphone applications [23]. Because of its importance, power modeling has been attracting an increasing amount of research effort [24–29]. In this project, we in particular study how to build accurate models for CPU power consumption in multicore smartphones.

Existing CPU power modeling approaches for smartphones assume CPU operating frequency and CPU utilization are the only major factors that affect CPU power consumption [24–26]. However, we find that this assumption does not hold with multicore CPUs in modern smartphones: under the same frequency and CPU utilization, two workloads with different CPU usage patterns could consume significantly different amounts of energy. Our experiments show that the difference can reach 50% in a quad-core Google Nexus 4 smartphone. Therefore, existing smartphone CPU power models are not suited for multicore smartphones. Our measurements indicate that the existing CPU power models give an estimation error as high as 34% on modern multicore smartphones. Moreover, the estimation accuracy of existing models is also notably unstable: the same CPU power model could generate an estimation variation larger than 30% for the different types of workloads.

The root cause of the estimation inaccuracy and instability comes from multiple newly introduced CPU *idle power states*, which consume markedly different amounts of power in multicore CPUs. We have carefully analyzed the impacts of idle power states on CPU power consumption, and developed a new CPU power-modeling method that treats CPU idle power states as a new major factor of CPU

power modeling [30]. As a result, the new modeling method is able to significantly improve power estimation accuracy and stability. To the best of our knowledge, our work is the first to target accurate CPU power modeling for multicore smartphone CPUs.

1.2.5 Enabling lightweight and privacy preserving location proofs for location based service applications

Location Based Service (LBS) application is a new and major category of applications in smartphone applications. An important problem of providing trustworthy LBS applications in smartphones is to enable efficient location proof schemes. A location proof scheme allows LBS providers to verify if users' location claims are in accordance with their actual location history, and to exclude those falsified ones. For example, in location based online social networks, such as Facebook and Foursquared, users can post comments and ratings about the venues they visited. Without proper location proof schemes, malicious users can comment on any places without actually visiting them. Another example is in today's Intelligent Transportation Systems (ITS), a popular category of applications is that vehicles report information about the transportation system elements (e.g., drivers and road conditions) to the ITS system for services like real time traffic control and roads maintenance [31, 32]. Successes of recent research projects on vehicle-based data sensing and collection [33–35] have bolstered such ITS data collection applications. However, before accepting data about a location reported by a vehicle, ITS operators need to verify if the vehicle visited the location at the time indicated in the reported data. Failing to do so will allow malicious users to launch an attack to the ITS system by reporting fake information about places where he did not actually visit. The damages of the attack are particularly serious, since the attacker can report fake information about numerous places by just clicking mouse at home. Therefore, studying how to provide efficient location reporting

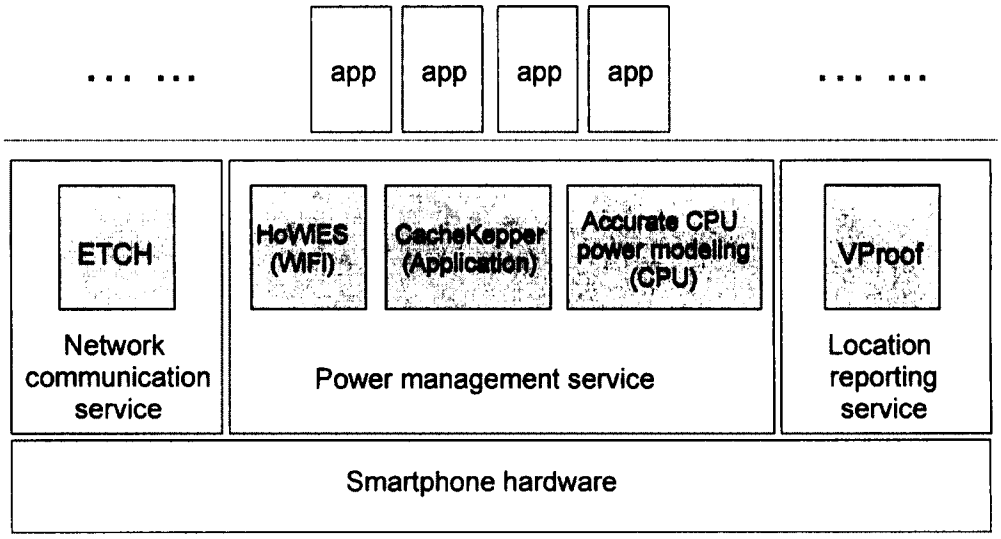


Figure 1: Summary of the problems.

service is important to enable efficient LBS applications in smartphones.

We propose *VProof*, a vehicle location proof scheme that enables users on vehicles to prove their location claims match their historical locations [36, 37]. With VProof, applications construct their location proofs by simply extracting relevant contents from the packets received from roadside units. Our scheme is lightweight, since there is no communication required for a prover to obtain a location proof. Our scheme also well preserves users' privacy, as we do not put any information that can be related to a user's ID in a location proof. We have implemented a prototype VProof system and evaluated it with extensive real-world experiments. Our evaluation results show that VProof is able to reliably prove vehicle's locations without leaking any user privacy. Although our current design and evaluation are performed on a vehicular basis, VProof can be extended to other mobile scenarios (e.g., waling and bicycling) with simple modifications.

1.3 Summary and organization

This dissertation proposes to improve efficiency of smartphone applications by improving the three fundamental services in smartphones: the wireless communi-

cation service, the power management service, and the location reporting service. We have conducted five projects for this purpose (summarized in Figure 1).

1. The first project (ETCH) improves the wireless communication service by achieving efficient rendezvous for cognitive radio communications.

2. The second project (HoWiES) improves the power management service from the perspective of WiFi communications.

3. The third project (CacheKeeper) improves the power management service from the perspective of applications.

4. The fourth project (accurate CPU power modeling) improves the power management service from the perspective of smartphone CPUs.

5. The last project (VProof) improves the location reporting service by providing a lightweight and privacy-preserving location proof scheme.

The rest of the dissertation is organized as follows. In Chapter 2, we present the details of ETCH. In Chapter 3, Chapter 4, and Chapter 5, we describe the three projects for improving the power management service respectively: HoWiES, CacheKeeper, and the accurate CPU power modeling method for multicore smartphones. In Chapter 6, we present the details of VProof. Finally, we conclude the dissertation and present our vision for future work in Chapter 7.

2 ETCH: Efficient Channel Hopping Based Communication Rendezvous for Cognitive Radio Communications

Wireless communication is important to mobile application in smartphones. We are facing an increasingly severe wireless spectrum scarcity problem, where the unlicensed bands are getting overcrowded. Cognitive radio is a promising technology to solve the problem, and thus is expected to shipped with future smartphones. We studied how to achieve efficient communication rendezvous, which is a critical step of establishing communication in cognitive radio networks.

2.1 Background and related work

2.1.1 Background

Communication rendezvous in cognitive radio (or dynamic spectrum access, DSA for short) networks is the process of establishing a control channel between two network nodes, over which they can exchange essential control information, before the pair of nodes can communication with each other. The common control channel approach, where a well-known channel is designated as control channel

for all nodes, is the most straightforward way to establish a control channel between a pair of DSA nodes. However, it suffers from the channel congestion problem and is vulnerable to jamming attacks [38]. Moreover, this approach cannot be applied in DSA networks because the control channel itself may be occupied by the primary user and hence become unavailable to the secondary users. The channel hopping approach, by contrast, increases control channel capacity and is immune to jamming attacks by utilizing multiple control channels. In this approach, all idle network nodes hop on a set of sequences of *rendezvous channels* (i.e., channels that are assigned for the purpose of control information exchange). When two nodes wishing to communicate hop to the same channel, this channel will serve as a control channel between the pair of nodes. The time that it takes for a pair of nodes to establish the control channel is called "*time-to-rendezvous*" or *TTR* for short.

To establish a control channel in DSA networks through channel hopping (abbreviation CH), every pair of nodes should have chance to rendezvous with each other periodically. In particular, due to the unique property of DSA networks that the channel availability is dynamic, the control channel established between any pair of nodes should equally likely be any one of the rendezvous channels. Otherwise, a pair of nodes would not be able to communicate if a primary user occupies the channels on which they rendezvous, even though there may still exist some other available channels to exchange the control information. QCH [7] is a recent control channel establishment protocol specifically designed for DSA networks. It utilizes the overlap property of quorums in a quorum system to develop CH sequences such that any two CH sequences are able to rendezvous periodically. Meanwhile, to accommodate the dynamics of the channel availability in DSA networks, QCH guarantees that any two nodes can meet each other as long as there are rendezvous channels not being occupied by primary users. While QCH is more suitable for DSA networks scenario and has better performances than exist-

ing CH-based multi-channel communication protocols, the following two concerns motivated us to explore for a better scheme.

First, in the scenario where global clock synchronization is available for DSA nodes to synchronize their channel hopping processes, QCH is only able to use one rendezvous channel as control channel in each hopping slot. This approach neglects the spectrum diversity, which is the most salient advantage brought by the DSA technique, in control channel establishment, and thus will potentially lead to severe traffic collision in a high probability. We propose SYNC-ETCH, a synchronous ETCH protocol, which efficiently exploits the spectrum diversity in a way that *every* rendezvous channel can serve as a control channel in *each* hopping slot. In SYNC-ETCH, while achieving the same goal, two CH sequence construction algorithms are proposed: two-phase CH sequence construction [8] and single-phase sequence construction. These two algorithms are complementary in design. The single-phase algorithm can *guarantee* the satisfaction of the even use of rendezvous channels requirement, which states that all the rendezvous channels should have the same probability to appear in each constructed CH sequence. This requirement is important for CH based communication rendezvous protocols, since if a CH sequence is heavily using a certain rendezvous channel, the nodes hopping on this sequence will lose contact with other nodes if the heavily relied channel is taken away by the primary user. The constraint of the single-phase algorithm is that it requires the total amount of rendezvous channels to be an odd number. The two-phase CH sequence construction algorithm can be applied to DSA networks with an arbitrary number of rendezvous channel, but it tries (cannot guarantee) to satisfy the even use of rendezvous channel requirement. As will be showed later, both of the SYNC-ETCH CH sequence construction algorithms achieve the optimal average TTR under the premise that all the rendezvous channels should be utilized as control channels in every hopping slot.

Second, in the scenario where the channel hopping processes of different DSA nodes are not synchronized, QCH only guarantees two of the rendezvous channels to be used as control channels. This arrangement also does not take advantage of spectrum diversity in DSA networks, and may lead to communication outage when the primary users appear on the two channels. We propose ASYNC-ETCH, an asynchronous ETCH protocol, which solves the problems by using all rendezvous channels as control channels.

2.1.2 Related work

Channel hopping based rendezvous protocols in normal multi-channel wireless networks. SSCH [39] is a well known synchronous communication rendezvous protocol for IEEE 802.11 network. In SSCH, each node hops on a sequence of channels determined by multiple (channel, seed) pairs. The arrangement of the hopping sequence ensures that any two nodes have chance to rendezvous with very high probability. In very small chance that two nodes will never meet, a parity slot with fixed channel is introduced to allow the two nodes to communicate. CHMA [40] is another synchronous CH based rendezvous protocol. It directs all nodes to hop on a common channel sequence such that any two nodes can communicate while utilizing all the channels. These protocols for normal multi-channel wireless networks do not take into account some important properties of dynamic spectrum access (DSA) networks, e.g., dynamic availability of channels, and thus are not suitable to be applied in DSA networks. Moreover, these protocols do not consider exploiting spectrum diversity in each hopping slot. DSA networks usually have much more spectrum diversity than normal multi-channel wireless networks. Therefore, exploiting spectrum diversity in DSA networks will bring much more performance gain.

Spectrum sharing in DSA networks. DSA network research can be divided into the following areas [41]: spectrum sensing ([42–47]), spectrum management

([47,48]), spectrum mobility and spectrum sharing. Our work belongs to the area of spectrum sharing. In this area, techniques can be categorized into two classes based on network architecture. Techniques in the first class assume there is a centralized entity that is responsible for the spectrum allocation for all the secondary users in the network. DSAP [49] is a typical solution that belongs to this category. The second class of spectrum sharing techniques perform the sharing in a distributed manner. These techniques can be further divided into two groups based on the assumption about the existence of a common control channel. Techniques in the first group (e.g., DOSS [50]) use common control channels that are available to all secondary users for spectrum sharing information exchange. The second group of techniques, which do not rely on common control channel, allow DSA nodes rendezvous with each other and exchange spectrum sharing information in a dynamic manner. Among these techniques, some are based on channel hopping (detailed next) and some are not. HD-MAC [38] is a representative distributed technique that ensure rendezvous in DSA networks not based on channel hopping. In this scheme, secondary users self-organize into groups based on similarity of available channels. In each of the groups, a group control channel, elected by group members, is used to carry control information of the group nodes. A weakness of HD-MAC is that it relies on all-channel broadcast to spread spectrum availability information and control channel votes. Both sender and receiver of a broadcast message need to rotate on all their available channels to send or receive the message, which will take a long time in establishing the group control channel especially when the number of channels is high.

Channel hopping based rendezvous protocols in DSA networks. Our work, QCH [7], SeqR [51] and Jump-stay CH [52] are representative CH based rendezvous protocols in DSA networks. QCH [7] deal with communication rendezvous in both the synchronous scenario and the asynchronous scenario, while SeqR [51] and Jump-stay CH [52] only deal with the asynchronous scenario. Dif-

ferent from the previous work, ETCH focuses on exploiting the spectrum diversity, which is the most salient advantage of DSA networks, in designing communication rendezvous protocols (for both scenarios).

2.2 Problem Formulation

2.2.1 Problem setting

In a DSA network, there are N (orthogonal) licensed channels labeled as C_0, C_1, ..., C_{N-1} that can be used for control information exchange. In other words, there are N *rendezvous channels* in the DSA network. Any pair of nodes wishing to communicate with each other should first establish a control channel between them before data communications. We assume that there is no centralized entity that globally controls the allocation of communication channels, so the control channel establishment between a pair of nodes is executed in a distributed manner.

In a CH-based solution, idle nodes[1] **periodically** hop on (i.e., switch their working channel according to) a *CH sequence*, which is a sequence of rendezvous channels. The time during which a node stays on a channel is defined as a *hopping slot*, which is notated as a (slot-index, channel) pair. Thus, a CH sequence S is notated as

$$S = \{(0, S[0]), (1, S[1]), ..., (i, S[i]), ..., (p-1, S[p-1])\},$$

where $i \in [0, p-1]$ is the index of a hopping slot, and $S[i] \in \{C_0, \cdots, C_{N-1}\}$ is the rendezvous channel assigned to the i-th slot of the sequence S. The time it takes for a node to hop through the entire CH sequence is called a *hopping period*. When two nodes hop to the same channel, they can hear from each other and that channel is established as their control channel. If more than two nodes

[1]Here idle nodes refer to nodes waiting to initiate a communication with other nodes and nodes waiting others to connect to them.

hop to the same rendezvous channel at the same time, they use existing collision avoidance mechanisms (e.g. RTS/CTS) or retransmission to establish pairwise control channels between them.

A CH-based solution should take account of the following requirements in its design.

- **Overlap requirement**. This requirement requires that any two CH sequences must overlap at a certain slot to ensure the rendezvous between the two nodes. Formally, given two CH sequences S_0 and S_1, they *overlap* if there exists a slot $(i, S_0[i]) \in S_0$ and a slot $(i, S_1[i]) \in S_1$ such that $S_0[i] = S_1[i]$. This slot is called an *overlapping slot* between S_0 and S_1, and the rendezvous channel $S_0[i]$ ($S_0[i] \in \{C_0, \cdots, C_{N-1}\}$) is called an *overlapping channel* between S_0 and S_1. If a rendezvous channel serves as an overlapping channel between a pair of CH sequences in the i-th slot, we say that *the rendezvous channel is utilized (as a control channel) in the i-th slot.*

- **Full utilization of rendezvous channels**. This requirement requires that any pair of nodes should be able to utilize every rendezvous channel as their control channel. This is to ensure the nodes have an opportunity to communicate with each other even if some of (but not all) the rendezvous channels are occupied by primary users.

- **Even use of rendezvous channels**. This requirement requires that all the rendezvous channels should have the same probability to appear in each CH sequence. If a CH sequence heavily relies on a certain channel (i.e., the channel is assigned to most of the slots of the CH sequence), nodes that hop on this CH sequence will lose contact with most of other nodes when the heavily relied channel is occupied by the primary user.

2.2.2 Metrics

We use the following three metrics in our numerical analysis for the proposed ETCH scheme.

- **Average rendezvous channel load.** This metric measures the average fraction of nodes that meet in the same rendezvous channel among all the nodes. Given a DSA network with M nodes and an average rendezvous channel load α $(0 < \alpha \leq 1)$, there are on average $M\alpha$ nodes rendezvous in the same channel. A light rendezvous channel load alleviates traffic collisions and increases the communication throughput.

- **Average time-to-rendezvous.** This is the average number of hopping slots that two nodes need to wait before they can rendezvous. A smaller average time-to-rendezvous (TTR) allows nodes to rendezvous and establish a communication link more quickly.

- **Rendezvous channel utilization ratio.** This is the ratio of the number of rendezvous channels that can be utilized as control channels in a hopping slot to the total number of rendezvous channels. It measures, in a given hopping slot, the extent that a communication rendezvous protocol utilizes the spectrum diversity in establishing control channels. A high rendezvous channel utilization ratio is helpful to reduce collision and improve the network capacity at the communication setup stage. This metric does not apply to the asynchronous case in which the hopping slot boundaries are not necessarily aligned.

We also use two other metrics, traffic throughput and actual time-to-rendezvous, to evaluate the practical performance of a communication rendezvous protocol. We will show that ETCH outperforms the existing solutions through mathematical analysis and simulations in §2.5 and §2.6 respectively.

2.2.3 Assumptions

We have the following assumptions regarding DSA networks and the node hardware.

- All the rendezvous channels are known to all the nodes. Information about rendezvous channels of a DSA network can be announced by regulation authorities such that all secondary users wishing to join the network will have this information.
- Each node is equipped with a single transceiver, which means a node cannot communicate in multiple channels at the same time. This assumption is in accordance with the ability of most commodity wireless devices.
- The channel switching overhead is negligible. This assumption is valid because most wireless hardware manufacturers claim that the channel switching delay is of the order of 80-90 μs [53]. This delay is negligible compared to the length of a slot in a hopping sequence, which is in the magnitude of 10ms.

2.3 SYNC-ETCH

SYNC-ETCH assumes that there exists a synchronization mechanism to achieve global clock synchronization among DSA nodes, so that two nodes wishing to communicate with each other can start channel hopping at the same time.

A newly joined node execute the SYNC-ETCH protocol in following two steps. In the first step, the node constructs a set of CH sequences by using either the *two-phase CH sequence construction* algorithm (§2.3.1) or the *single-phase CH sequence construction* algorithm (§2.3.2). The two-phase algorithm can be applied to scenarios with arbitrary numbers of rendezvous channels. It satisfies the overlap requirement in the first phase, and *tries* to fulfill the requirement of even

use of rendezvous channels in the second phase. The single-phase algorithm *guarantees* the satisfaction of both requirements in an integral design. Both of the algorithms achieve the optimal average TTR under the premise that all the rendezvous channels should be utilized as control channels in every hopping slot. The key design goal of both CH sequence construction algorithms is to fully utilize *all* the rendezvous channels in every hopping slot.

Theorem 1. *In a DSA network with N rendezvous channels, for any CH based synchronous communication rendezvous protocols where all the rendezvous channels are utilized in each hopping slot, the minimum number of hopping slots of each CH sequence is $2N-1$, and the average TTR is $\frac{2N-1}{2}$.*

Proof. To let all the N rendezvous channels be fully utilized in each CH time slot, we must arrange at least $2N$ CH sequences in a way that N pairs of CH sequences rendezvous at N different channels. We also must arrange at least $2N-1$ hopping slots for each of the $2N$ CH sequences to allow each sequence to rendezvous with the rest $2N-1$ CH sequences (for the overlap requirement). Considering that the rendezvous time of two randomly selected CH sequences (from the $2N$ sequences) is uniformly distributed between slot one and slot $2N-1$, the average TTR is $\frac{2N-1}{2}$. □

Theorem 1 reveals that, to fully utilize all the N rendezvous channels in each hopping slot, there are at least $2N-1$ hopping slots in each CH sequence. As we will show later, both CH sequence construction algorithms in SYNC-ETCH can achieve such optimal length of CH sequences.

In the second step, the node starts the CH sequence execution process (§2.3.3) in a way that the full utilization of rendezvous channels requirement is satisfied.

We introduce the CH sequence construction algorithms and the CH sequence execution algorithm in the rest of this section.

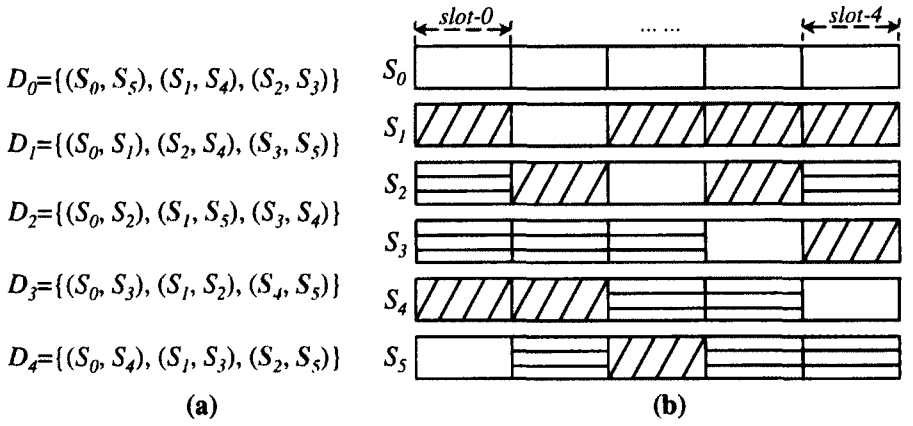


Figure 2: Phase 1 of the two-phase CH sequence construction - *rendezvous scheduling*. This figure shows the 5 rendezvous schedules (D_0 to D_4) of a DSA network with 3 rendezvous channels (i.e., $N = 3$). In this network, 6 CH sequences (S_0 to S_5) are constructed. Each CH sequence has 5 hopping slots. Rendezvous schedule $D_i (0 \leq i \leq 4)$ specifies how nodes following two different CH sequences rendezvous in the slot-i. For instance, in slot-0, the nodes hopping on CH sequence S_0 meet the nodes on S_5 in one of the 3 rendezvous channels, the nodes on S_1 meet the nodes on S_4 in a different rendezvous channel, and the nodes on S_2 meet the nodes on S_3 in the remaining channel.

2.3.1 Two-phase CH sequence construction

To simplify the presentation of the two-phase CH sequence construction algorithm, we first give an overview and an example of the construction process. Then, we provide the formalized algorithm.

An overview and an example

The two-phase CH sequence construction algorithm constructs a set of CH sequences in two phases. The *first* phase is called the *rendezvous scheduling* phase. In this phase, the algorithm creates a set of rendezvous schedules, each of which instruct how nodes with different CH sequences meet with each other in a hopping slot. Given a DSA network with N rendezvous channels, to fully utilize spectrum diversity, an ideal rendezvous schedule allows N pairs of nodes to rendezvous at N different channels in a hopping slot, which is equivalent to arrange for $2N$ CH sequences (each of which is used by one participating node) to overlap at different N channels in a slot. Meanwhile, the rendezvous schedules should

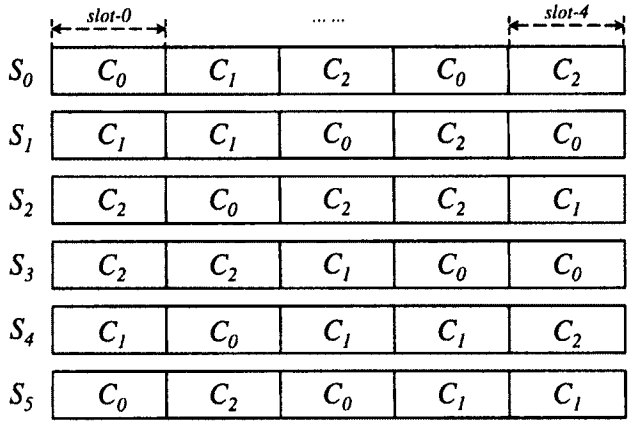


Figure 3: Phase 2 of the two-phase CH sequence construction - *rendezvous channel assignment*. This figure shows the 6 final CH sequences of a DSA network with 3 rendezvous channels C_0, C_1 and C_2 (i.e., $N = 3$). A greedy algorithm is used to assign the 3 rendezvous channels to each slot (slot-0 to slot-5) of all the 6 CH sequences (S_0 to S_5) based on the rendezvous schedules output by the rendezvous scheduling phase. For instance, nodes following the CH sequence S_0 hop on a sequence of channels $C_0 \rightarrow C_1 \rightarrow C_2 \rightarrow C_0 \rightarrow C_1$ periodically.

ensure the satisfaction of the overlap requirement, i.e., any pair of nodes hopping on different CH sequences can meet at least once within a hopping period.

The *second* phase is called the *rendezvous channel assignment* phase. In this phase, the algorithm fills the rendezvous channels in the $2N$ CH sequences based on the rendezvous schedules generated in the previous phase. This phase tries to satisfy the design requirement of even use of rendezvous channels by using a greedy algorithm. At the end of the rendezvous channel assignment phase, $2N$ CH sequences are constructed.

Figure 2 and Figure 3 illustrate an example of the two-phase CH sequence construction in a DSA network with three ($N = 3$) rendezvous channels. Figure 2(a) shows the five rendezvous schedules $D_0, D_1, ..., D_4$ generated in the rendezvous scheduling phase. Each rendezvous schedule corresponds one of the five $(2N - 1)$ hopping slots of the six $(2N)$ the CH sequences $S_0, S_1, ..., S_5$. As we can see, in each of the five hopping slots, six nodes (selecting different CH sequences) are supported to rendezvous in three different channels. For example, in the first slot (i.e., slot-0), where the rendezvous schedule D_0 is used to arrange

rendezvous, the node selecting CH sequence S_0 is arranged to rendezvous with the node selecting CH sequence S_5 on one rendezvous channel, while the node selecting S_1 meets with the node selecting S_4 on a different rendezvous channel, and the node selecting S_2 meets with the node selecting S_3 on the remaining rendezvous channel. Figure 2(b) shows the overall effect of how the six node selecting different CH sequences rendezvous in different hopping slot. In each hopping slot in Figure 2(b), a pair of nodes whose CH sequences have the same type of shade will meet on the same rendezvous channel. Please note that the detailed arrangement about rendezvous channels on which pairs of nodes rendezvous has not yet been determined in this phase, and is left to the next phase. As will be presented shortly, our algorithm schedules the $2N$ CH sequences to meet in the N different rendezvous channels in a hopping slot as follows. It selects $2N-2$ out of the first $2N-1$ CH sequences (i.e., $S_0, \cdots, S_{2N-2}$) to form $N-1$ CH sequences pairs, where each sequence is scheduled to meet the other sequence from the same pair in the slot-sl, such that the index sum of each pair of CH sequences is congruent to sl modulo $2N-1$. The remaining CH sequence (within $S_0, \cdots, S_{2N-2}$) and S_{2N-1} form the last pair of CH sequences (shown in blue color in Fig 2(a)) that are scheduled to meet in the slot-sl.

Figure 3 shows an example of rendezvous channel assignment once the scheduling is determined. From the example we can see that all the rendezvous channels are utilized for communication in each of the hopping slots, and that each rendezvous channel appears in each of the CH sequences with roughly the same probability.

Phase 1: rendezvous scheduling

We now formalize the problem of rendezvous scheduling, the first phase of the two-phase CH sequence construction process, as follows. Given a set of $2N$ CH sequences $U = \{S_0, S_1 \cdots, S_{2N-1}\}$, $D_{sl} = \{d_0, d_1, \cdots, d_{N-1}\}$ is called a *ren-*

Algorithm 1: Rendezvous Scheduling

Data: $U = \{S_0, \cdots, S_{2N-1}\}$: a set of $2N$ empty CH sequences, each of which has $2N-1$ slots;
Result: $\mathcal{D} = \{D_0, D_1, \cdots, D_{2N-2}\}$: $2N-1$ different rendezvous schedules of U.

```
Initialize D_0, D_1, ..., D_{2N-2} to be empty;
for sl ← 0 to 2N − 2 do
    V ← U\{S_{2N-1}};
    for i ← 0 to N − 1 do
        a ← the smallest subscript in V;
        if a ≤ sl then
            b ← sl − a;
        else
            b ← 2N − 1 + sl − a;
        if a == b then
            b ← 2N − 1;
        d_i ← {S_a, S_b};
        D_sl ← D_sl ∪ {d_i};
        V ← V\{S_a, S_b};
return D_0, D_1, ..., D_{2N-2};
```

dezvous schedule for the hopping slot indexed in sl if $\bigcup D_{sl} = d_0 \cup d_1 \cup \cdots \cup d_{N-1} = U$, where $d_i = \{S_a, S_b\}$ $(0 \leq i \leq N-1)$ is a pair of CH sequences that are scheduled to rendezvous in the slot-sl.

According to theorem 1, the optimal rendezvous scheduling algorithm must construct $2N-1$ different rendezvous schedules, each of which corresponds to a hopping slot, such that each CH sequence is able to rendezvous with all the other $2N-1$ CH sequences in $2N-1$ hopping slots. SYNC-ETCH uses Algorithm 1 to construct the schedules.

In Algorithm 1, rendezvous schedule D_{sl} $(0 \leq sl \leq 2N-2)$, which is the rendezvous schedule for the slot-sl, is constructed as follows. Within the CH sequences set $V = \{S_0, \cdots, S_{2N-2}\}$, S_a and S_b are scheduled to rendezvous in the slot-sl (i.e., $\{S_a, S_b\} \in D_{sl}$) if $a + b \equiv sl(\text{mod}(2N-1))$ and $a \neq b$. For the CH sequence $S_a \in V$ that satisfies $2a \equiv sl(\text{mod}(2N-1))$, it is scheduled to rendezvous with the CH sequence S_{2N-1} in the slot-sl (i.e., $\{S_a, S_{2N-1}\} \in D_{sl}$). We prove the correctness of Algorithm 1 as follows.

Theorem 2. *Algorithm 1 constructs $2N-1$ rendezvous schedules of U, and all these $2N-1$ rendezvous schedules are different.*

Proof. In order to prove Algorithm 1 constructs $2N-1$ rendezvous schedules, we need to prove given an integer sl $(0 \leq sl \leq 2N-2)$, D_{sl} is a rendezvous schedule of U. To prove this, we need to prove

(1) there is only a number $x \in [0, 2N-2]$ such that $2x \equiv sl \pmod{(2N-1)}$, and

(2) $\forall\ a, b, c, d \in [0, 2N-2]$ that satisfy $a+b \equiv sl \pmod{(2N-1)}$ and $c+d \equiv sl \pmod{(2N-1)}$, if $a \neq c$ then $b \neq d$.

By proving (1) we can guarantee that the CH sequence S_{2N-1} only exists in only a CH sequence pair d_i $(0 \leq i \leq N-1)$ within rendezvous schedule D_{sl}. From (1), (2) and the strategy that we always choose the first CH sequence of d_i $(0 \leq i \leq N-1)$ from a set of CH sequences that have never been chosen (i.e. set V in Algorithm 1)(line 5), we can ensure that $\bigcup D_{sl} = d_0 \cup d_1 \cup \cdots \cup d_{N-1} = U$ (i.e. D_{sl} is a rendezvous schedule of U).

We prove both (1) and (2) by contradiction. For (1), suppose there are two different number m and n that satisfy $0 \leq m < n \leq 2N-2$, $2m \equiv sl \pmod{(2N-1)}$ and $2n \equiv sl \pmod{(2N-1)}$, then we can have $2m = sl$ and $2n = 2N-1+sl$. A contradiction is found that sl is an even number because $2m = sl$, and sl is an odd number because $2n = 2N-1+sl$. For (2), without loss of generality, we suppose $a < c$. If $b = d$, then we have $a+b = sl$ and $c+d = 2N-1+sl$. By subtracting these two equations we get $c-a = 2N-1$ which is impossible because $0 \leq a < c \leq 2N-2$.

In order to prove $\forall p, q \in [0, 2N-2]$ $(p \neq q)$, rendezvous schedule D_p and schedule D_q are different, we need to prove $\forall d_i \in D_p (0 \leq i \leq N-1)$ and $\forall d_j \in D_q (0 \leq j \leq N-1)$, $d_i \neq d_j$. We prove this by contradiction. Suppose there exist $d_i \in D_p$ and $d_j \in D_q$ such that $d_i = d_j$, which means d_i and d_j contain the same pair of CH sequences. Suppose these two sequences are S_u and S_v, where $0 \leq u, v \leq 2N-1$. Then we have $u+v \equiv p \pmod{(2N-1)}$ and $u+v \equiv$

$q \pmod{(2N-1)}$, where $p, q \in [0, 2N-2]$ and $p \neq q$, which is impossible. □

Phase 2: rendezvous channel assignment

In the second phase of the two-phase CH sequence construction process, we assign rendezvous channels to each of the $2N$ CH sequences according to the rendezvous schedules generated in the previous phase. The goal of the rendezvous channel assignment phase is two-fold. *First*, to fully exploit the frequency diversity of a DSA network in establishing control channels, *all* the rendezvous channels should be utilized in each hopping slot. *Second*, the assignment tries to satisfy the even use of rendezvous channels requirement presented in §2.2.1 by an arrangement that allows each rendezvous channel to have a roughly equal probability to appear in each CH sequence.

We employ a greedy algorithm (shown in Algorithm 2) to achieve the goals of rendezvous channel assignment. In Algorithm 2, rendezvous channels are assigned to CH sequences round by round (lines 3-18). In the sl-th $(0 \leq sl \leq 2N-2)$ round, the rendezvous channels are assigned to the slot-sl of all the CH sequences based on the slot's rendezvous schedule, D_{sl}, constructed in the previous phase. For each hopping slot, the algorithm needs to guarantee that every rendezvous channel is assigned to a pair of CH sequences (to achieve the first goal of rendezvous channel assignment). To keep track of the channel assignment for each slot, the variable $slotOC$ is used to record the outstanding rendezvous channels of the current slot, i.e., the rendezvous channels that have not been assigned to the slot. At the beginning of each round of channel assignment, $slotOC$ is reset to the whole set of rendezvous channels (line 4). The algorithm also tries to make all the rendezvous channels appear in each CH sequence with a roughly equal probability (to achieve the second goal of rendezvous channel assignment). To keep track of the channel assignment for each CH sequence, the variable $seqOC[i]$ $(0 \leq i \leq 2N-1)$ is used to record the outstanding rendezvous

Algorithm 2: Rendezvous Channel Assignment

Data: $\mathcal{C} = \{C_0, C_1, \cdots, C_{N-1}\}$: N rend. channels; $\mathcal{D} = \{D_0, D_1, \cdots, D_{2N-2}\}$: $2N-1$ rendezvous schedules returned by Algorithm 1.

Result: $S_0, S_1, \cdots, S_{2N-1}$: $2N$ final CH sequences.

1 **for** $i \leftarrow 0$ **to** $2N-1$ **do**
2 | $seqOC[i] \leftarrow \{C_0, C_1, \cdots, C_{N-1}\}$; */*Initializing the outstanding channels of the CH sequence S_i.*/*
3 **for** $sl \leftarrow 0$ **to** $2N-2$ **do**
4 | $slotOC \leftarrow \{C_0, C_1, \cdots, C_{N-1}\}$; */*Initializing the outstanding channels of the slot-sl.*/*
5 | **for** $n \leftarrow 0$ **to** $N-1$ **do**
6 | | Mark CH sequence pair $d_n \in D_{sl}$ as *unassigned*;
7 | **while** $slotOC \neq \phi$ **do**
8 | | Pick $d_n = \{S_i, S_j\}$ from the unassigned CH sequence pairs in D_{sl} such that $seqOC[i] + seqOC[j]$ is the greatest (if multiple choices exist, pick the pair that contains the CH sequence with the smallest index);
9 | | $k \leftarrow seqOC[i] \geq seqOC[j]\ ?\ i : j$;
10 | | **if** $slotOC \cap seqOC[k] \neq \phi$ **then**
11 | | | $c \leftarrow$ the channel in $slotOC \cap seqOC[k]$ with the smallest index;
12 | | | $seqOC[k] \leftarrow seqOC[k] \backslash \{c\}$;
13 | | **else**
14 | | | $c \leftarrow$ the channel in $slotOC$ that appears the fewest times in S_k (if multiple choices exist, pick the one with the smallest index);
15 | | $S_i \leftarrow S_i \cup (sl, c)$;
16 | | $S_j \leftarrow S_j \cup (sl, c)$;
17 | | Mark $d_n = \{S_i, S_j\}$ as *assigned*;
18 | | $slotOC \leftarrow slotOC \backslash \{c\}$;
19 **return** $S_0, S_1, \cdots, S_{2N-1}$;

channels of the CH sequence S_i, i.e., the rendezvous channels that have not been assigned to the CH sequence S_i. The variables $seqOC[i]$ are initialized to the whole set of rendezvous channels (lines 1-2).

Before the sl-th round of rendezvous channel assignment (i.e., the round that assign channels to slot-sl of all the CH sequences), all the CH sequence pairs in D_{sl} are initially marked as "unassigned" (lines 5-6). In the sl-th round of rendezvous channel assignment, the algorithm checks all the unassigned CH sequence pairs in D_{sl}, and selects the pair $\{S_i, S_j\}$ such that the sum of outstanding channels of S_i and S_j is *greatest* compared to other unscheduled CH sequence pairs in D_{sl}. If there are multiple pairs that produce the same greatest outstand-

ing channels sum, the pair that contains the smallest indexed CH sequence is selected (line 8). Then the algorithm chooses a rendezvous channel to assign to the slot-sl of both S_i and S_j (lines 9-16). This rendezvous channel is selected as follows. Within the CH sequences S_i and S_j, the one with more outstanding channels is notated as S_k (line 9). The rendezvous channel is first selected from the intersection of the slot-sl's outstanding channels (recorded in $slotOC$) and S_k's outstanding channels (recorded in $seqOC[k]$) (line 11). If the intersection is empty, the channel is selected from the slot-sl's outstanding channel that appears *fewest* times in S_k (line 14). Then this rendezvous channel is assigned to the slot-sl of both CH sequences of S_i and S_j (lines 15-16), and the CH sequence pair $\{S_i, S_j\}$ is marked as "assigned" (line 17). The selected rendezvous channel is removed from the current slot's outstanding channels set (line 18). It is also removed from S_k's outstanding channels if has not been assigned to S_k before the assignment (line 12).

Figure 3 shows the result of rendezvous channel assignment in a DSA network with 3 rendezvous channels, C_0, C_1 and C_2. CH sequences S_0 to S_5 are the final CH sequences constructed by the two-phase CH sequence construction process. From the example we can see that all the rendezvous channels are utilized for communication in each of the hopping slots, and that each rendezvous channel appears in each of the CH sequences with roughly the same probability.

2.3.2 Single-phase CH sequence construction

Similar to the presentation of the two-phase CH sequence construction process, we provide an overview and an example of the single-phase CH sequence construction process, followed by the formalized algorithm.

An overview and an example

In a DSA network with N rendezvous channels, similar to the two-phase algorithm, the single-phase CH sequence construction algorithm constructs $2N$ CH sequences, each of which has $2N-1$ hopping slots, such that the following two requirements are satisfied. *First*, every CH sequence meets with all the other $2N-1$ CH sequences each at a time in a hopping slot. *Second*, all the rendezvous channels are utilized for CH sequence rendezvous in every hopping slot. The improvement of the single-phase algorithm over the two-phase algorithm is that it can guarantee to satisfy a *third* requirement that every rendezvous channel has the same probability to appear in each CH sequence (i.e., the even use of rendezvous channels requirement). For instance, in a DSA network with three rendezvous channels, the even use of rendezvous channels requirement expects there are two rendezvous channels appearing twice and the remaining channel appearing once in the five hopping slots of *every* CH sequence. However, in the six CH sequences constructed by the two-phase algorithm (shown in Figure 3), channel C_2 and C_1 appear three times in the CH sequence S_2 and S_4 respectively. By contrast, the single-phase algorithm can guarantee the even use of rendezvous channels requirement.

The single-phase CH sequence construction algorithm views the rendezvous among the CH sequences within a hopping period as a colored graph G. To satisfy the three requirements above, the colored graph G should have the following properties. *First*, there are $2N$ vertices in G:

$$|V(G)| = 2N, \tag{1}$$

where $V(G)$ denotes the vertex set of the graph G. Each vertex corresponds to one of the $2N$ CH sequences. *Second*, the edges in G have N different colors:

$$C(G) = \{c_0, ..., c_{N-1}\}, \tag{2}$$

where $C(G)$ denotes the color set on edges in the graph G. Each color corresponds to a rendezvous channel. If two CH sequences rendezvous in a certain channel, the corresponding pair of vertices in G are connected by an edge with the corresponding color. *Third*, since every CH sequence should rendezvous with all the other $2N-1$ sequences exactly once in a hopping period, the graph G should satisfy

$$\forall v \in V(G), d(v) = 2N - 1, \tag{3}$$

where $d(v)$ denotes the degree of vertex v. In other words, the graph G should be a $2N$-vertex complete graph K_{2N}. *Fourth*, since all the rendezvous channels should appear in every CH sequence, the color degree of each vertex, which is the number of colors on the edges incident to the vertex, should be N:

$$\forall v \in V(G), \delta(v) = N, \tag{4}$$

where $\delta(v)$ is the number of colors on edges incident to the vertex v. *Fifth*, since each of the N rendezvous channel should have the same probability to appear in every CH sequence (i.e., the even use of rendezvous channels requirement), among the N different colors on the $2N-1$ edges incident to a vertex v, there should be one color to appear once and the remaining $N-1$ colors to appear twice, which is the best scenario satisfying the even use of rendezvous channels requirement:

$$\forall v \in V(G)\ (\exists c_i \in C(G)\ (\delta_{c_i}(v) = 1 \&\& \delta_{c_j}(v) = 2, \forall j \in [0, N-1], j \neq i)), \tag{5}$$

where $\delta_{c_i}(v)$ is the number of edges colored with c_i that are incident to the vertex v. Figure 5(a) shows an example of the graph G for a DSA network with 5 rendezvous channels (i.e., $N = 5$). The graph G in the example is a 5-colored 10-vertex complete graph K_{10}. In this graph, each of the 5 colors has an even probability to appear on the 9 edges that are incident to each vertex (i.e., one color appears once and each of the rest four colors appear twice), which is the best case of satisfying the even use of rendezvous channels requirement.

The graph G with the properties (1) to (5) tells how each of the $2N$ CH sequence meets with each other in the N rendezvous channels *within a hopping period*. The single-phase CH sequence construction algorithm needs to further specify how the CH sequences rendezvous with each other in each of the $2N-1$ hopping slots. To fully exploit the spectrum diversity, our algorithm ensures that all the rendezvous channels can be utilized as control channel in every hopping slot. This is achieved by decomposing the graph G into $2N-1$ *different perfect rainbow matchings*, each of which instructs how the $2N$ CH sequences rendezvous in a hopping slot. In graph theory, a *matching* in a graph G is a set of edges of G without common vertices, a *perfect matching* in G is a matching that covers all the vertices of the graph G [54], and a *rainbow matching* in G is a matching where edges have distinct colors [55]. Therefore, in our case, a *perfect rainbow matching* (notated as PRM) in a N-colored $2N$-vertex complete graph G is an edge set that contains N disjoint edges of G colored with the N distinct colors:

$$PRM = \{\hat{E} \mid V(\hat{E}) = V(G) \;\&\&\; \forall e_i \in \hat{E}\ (\forall v \in V(e_i)\ (v \notin V(e_j), \forall e_j \in \hat{E}, j \neq i)) \\ \&\&\; \forall e_i, e_j \in \hat{E},\ i \neq j\ (C(e_i) \neq C(e_j))\} \quad (6)$$

where $V(\hat{E})$ denotes the set of vertices of the edge set $\hat{E}$, $V(e_i)$ denotes the two vertices on the edge e_i, and $C(e_i)$ denotes the color on the edge e_i. Given two perfect rainbow matching PRM_i and PRM_j, they are different if and only if the there is no common edges in them:

$$PRM_i \text{ and } PRM_j \text{ are different } \texttt{iff } \forall e_i \in E(PRM_i), e_j \in E(PRM_j)\ (e_i \neq e_j), \quad (7)$$

where $E(PRM_i)$ denotes the edge set in the perfect rainbow matching PRM_i. In our example, the 5-colored 10-vertex complete graph G shown in Figure 5(a) can be decomposed into 9 different *PRM*s shown in Figure 4(1), Figure 4(6a-1) to (6a-4), and Figure 4 (6b-1) to (6b-4) respectively. Each of these *PRM*s is the rendezvous schedule for one hopping slot within a hopping period. The final

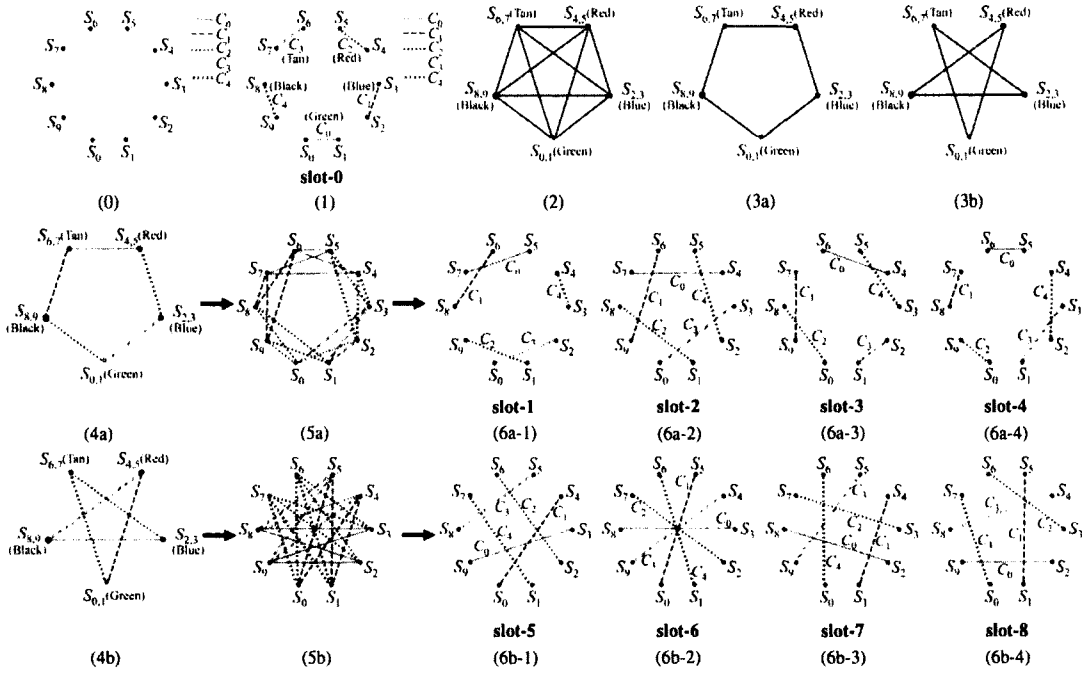


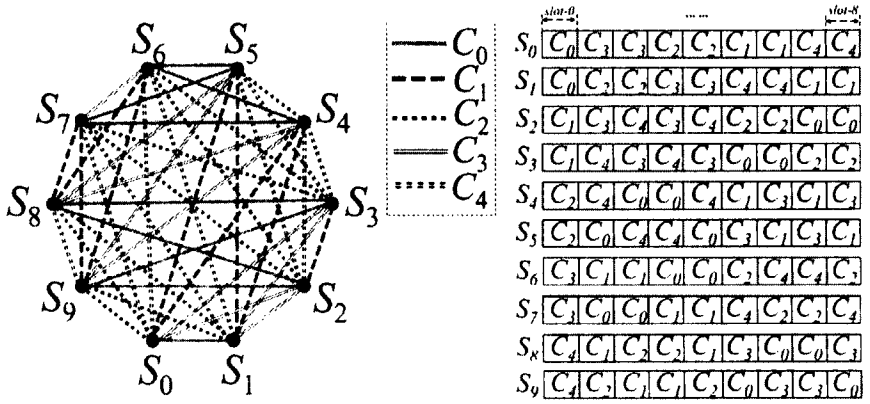
Figure 4: The six steps of the intuitive description of the single-phase CH sequence construction algorithm for a DSA network with 5 rendezvous channels (i.e., $N = 5$). (0) is the initial state of the graph G; (1) shows how the the first PRM is formed in the step #1; (2) shows how the graph G is shrank to K_5 in the step #2 based on the first PRM; (3a) and (3b) are the two 2-factors of K_5 obtained in the step #3; (4a) and (4b) show the rainbow-coloring of the two 2-factors of K_5 in the step #4; (5a) and (5b) show how the two rainbow-colored 2-factors of K_5 are expanded back to the two 4-factors in K_{10} in the step #5; (6a-1) to (6a-4) and (6b-1) to (6b-4) are respectively the final 4 PRMs decomposed from the two rainbow-colored 4-factors of K_{10}.

CH sequences (shown in Figure 5(b)) are constructed based on these 9 different *PRM*s.

In the following, we will show that, for a DSA network with N rendezvous channels, where N is an odd number greater than two, our single-phase CH sequence construction algorithm can form a graph G with the properties of (1) to (5), and decompose the graph G into $2N - 1$ different perfect rainbow matchings (i.e., properties of (6) and (7)).

An intuitive description of the algorithm

For a DSA network with N rendezvous channels, where N is an odd number greater than two, the single-phase CH sequence construction algorithm constructs



(a) (b)

Figure 5: For a DSA network with 5 rendezvous channels (i.e., $N = 5$), (a) is the N-colored $2N$-vertex complete graph G that shows how the $2N$ CH sequence rendezvous with each other within a hopping period, and (b) shows the final $2N$ CH sequences.

the $2N$ CH sequences as follows.

The initial state: We treat the $2N$ CH sequences to be constructed as $2N$ vertices in a graph G, and assign different colors to each of the N rendezvous channels. Initially, the graph G only contains the $2N$ vertices and no edges. Our goal is to add colored edges to G to make it an N-colored $2N$-vertex complete graph K_{2N} with the properties (1) to (5), and then decompose the K_{2N} into $2N-1$ different perfect rainbow matchings (i.e., properties (6) and (7)).

In the following, we notate the $2N$ CH sequences as $S_0 \cdots S_{2N-1}$, and the N rendezvous channels as $C_0, \cdots, C_{N-1}$. We will take a DSA networks with 5 rendezvous channels (i.e., $N = 5$) as an example throughout this subsection. Figure 4(0) shows the initial state of the graph G in the example.

Step #1 - forming the first perfect rainbow matching of K_{2N}: In the first step, we form the first PRM by connecting vertex S_i with vertex S_{i+1} by using an edge with the color of channel $C_{\frac{i}{2}}$, where i is an even number in the range of $[0, 2N-1]$. The PRM tells how the $2N$ CH sequences rendezvous with each other in the hopping slot-0. In our example, Figure 4(1) shows the state of the graph G after the step #1 is applied. It is a PRM of a $2N$-vertex complete graph

Algorithm 3: 2-factorization of complete graph K_N

Data: K_N formed after step #2, the N vertices of K_N are $\{S_{0,1}, \cdots, S_{2N-2,2N-1}\}$;
Result: $\frac{N-1}{2}$ 2-factors of K_N: $TF_1, \cdots, TF_{\frac{N-1}{2}}$.

1 **for** $d \leftarrow 1$ **to** $\frac{N-1}{2}$ **do**
2 | $TF_d \leftarrow \phi$;
3 | **for** *each edge* $(S_{i,i+1}, S_{j,j+1}) \in E(K_N)$ **do**
4 | | **if** $\frac{|i-j| \ \% \ N}{2} == d$ **then**
5 | | | Add edge $(S_{i,i+1}, S_{j,j+1})$ and its vertices to TF_d;
6 **return** $TF_1, \cdots, TF_{\frac{N-1}{2}}$;

K_{2N}, which specifies that in the hopping slot-0, CH sequences S_0 and S_1, S_2 and S_3, S_4 and S_5, S_6 and S_7, S_7 and S_8 rendezvous in channel C_0 to C_4 respectively.

Step #2 - shrinking the $2N$ vertex graph G to K_N: In the second step, we shrink the $2N$-vertex graph G to an N-vertex N-colored complete graph K_N as follows. First, we combine every connected vertex pair in the first PRM of K_{2N} (i.e., $\{S_i, S_{i+1}\}$, where $i = 2a, a \in [0, N-1]$) into a new vertex (notated as $S_{i,i+1}$), and connect the new vertices to each other to form a N-vertex complete graph K_N. Second, we give each vertex in K_N the color of the edge connecting the corresponding vertex pair in the first PRM of K_{2N}. Figure 4(2) shows the state of the graph G after the step #2 is applied.

Step #3 - decomposing K_N into $\frac{N-1}{2}$ different 2-factors: In graph theory, a 2-factor of a graph G is spanning subgraph of G, where the degree of each vertex in the subgraph is 2. Algorithm 3 decomposes K_N into $\frac{N-1}{2}$ 2-factors. In the algorithm, each edge of K_N and its vertices are put into $\frac{N-1}{2}$ graphs, $TF_1, \cdots, TF_{\frac{N-1}{2}}$, depending on the subscript difference between the two vertices: given an edge $e = (S_{i,i+1}, S_{j,j+1}) \in E(K_N)$, where $i < j$, the edge e and its vertices are put into the graph TF_d $(1 \leq d \leq \frac{N-1}{2})$ if either $\frac{j-i}{2}$ or $\frac{i+2N-j}{2}$ equals to d (line 5). We prove in Theorem 3 that each graph TF_d $(1 \leq d \leq \frac{N-1}{2})$ decomposed from K_N using Algorithm 3 is a 2-factor of the complete graph K_N. It is obvious that the $\frac{N-1}{2}$ 2-factors are different, since each edge of K_N can only belong to one 2-factor. In

Algorithm 4: Rainbow-coloring a 2-factor of K_N

Data: A 2-factor TF obtained in step #3;
Result: Rainbow-coloring the 2-factor such that every edge's color is different from the colors of its two vertices.

1 **for** *each edge* $e = (S_{i,i+1}, S_{j,j+1}) \in TF$ **do**
2 **if** $\frac{i+j}{2}$ *is an even number* **then**
3 Put the color of vertex $S_{\frac{i+j}{2},\frac{i+j}{2}+1}$ on the edge e;
4 **else**
5 Put the color of vertex $S_{\frac{i+j+2N}{2}\%2N,\frac{i+j+2N}{2}\%2N+1}$ on the edge e;

our example, Figure 4(3a) and Figure 4(3b) are the two 2-factors of the complete graph K_5 obtained after the step #2. They are both Hamiltonian cycles of K_5.

Theorem 3. *Each graph* TF_d $(1 \leq d \leq \frac{N-1}{2})$ *decomposed from* K_N *using Algorithm 3 is a 2-factor of the complete graph* K_N.

Proof. In the algorithm, among all the edges that are incident to *each* $S_{2i,2i+1} \in V(K_N)$ $(0 \leq i \leq N-1)$, only the edge $(S_{2i,2i+1}, S_{(2(i+d))\%2N,(2(i+d)+1)\%2N})$ and the edge $(S_{2i,2i+1}, S_{(2(i-d))\%2N,(2(i-d)+1)\%2N})$ and their associated vertices are added to the graph TF_d. Since $d \neq 0$ and $d \neq \frac{N}{2}$, we have $(2(i+d))\%2N \neq (2(i-d))\%2N$, which means the two edges added to TF_d are different. Therefore, $V(TF_d)$ equals to $V(K_N)$, and the degree of each vertex in TF_d is 2. Thus, TF_d is a 2-factor of the complete graph K_N.

Simple calculation can easily confirm that the 2-factor TF_d is either a Hamiltonian cycle of K_N if d and N is coprime, or $GCD(N,d)$ disjoint $\frac{N}{GCD(N,d)}$-cycles (i.e., cycles with $\frac{N}{GCD(N,d)}$ edges), where $GCD(N,d)$ is the greatest common divisor of N and d, otherwise. □

Step #4 - rainbow-coloring 2-factors of K_N: In the fourth step, we color each of K_N's 2-factors using all the N colors such that each edge will have a different color, which is a process called "rainbow-coloring".

Algorithm 4 shows the rainbow-coloring algorithm. In this algorithm, the color to put on a edge is the color of another vertex that is different from the two

vertices of the edge. Specifically, the color to put on the edge $(S_{i,i+1}, S_{j,j+1})$ is the color of either the vertex $S_{\frac{i+j}{2},\frac{i+j}{2}+1}$ (if $\frac{i+j}{2}$ is an even number) or the vertex $S_{\frac{i+j+2N}{2}\%2N,\frac{i+j+2N}{2}\%2N+1}$ (if $\frac{i+j}{2}$ is an odd number). We have the following two theorems about the rainbow-coloring algorithm.

Theorem 4. *The coloring process is a rainbow-coloring process (i.e., after the coloring process, the colors on the N edges of each 2-factor are different).*

Proof. Prove by contradiction. Suppose there exist two different edges $e_1 = (S_{i,i+1}, S_{j,j+1})$ and $e_2 = (S_{m,m+1}, S_{n,n+1})$ in the the 2-factor TF_d ($d \in [1, \frac{N-1}{2}]$) that have the same color. Since e_1 and e_2 are not the same, without loss of generality, let us assume $i \neq m$, $i > j$ and $m > n$.

Since e_1 and e_2 have the same color, they correspond to the same vertex in K_N. Without loss of generality, let us assume $\frac{i+j}{2}$ is an even number, then we have $\frac{i+j}{2} = \frac{m+n}{2}$ $(*)$.

Meanwhile, since e_1 and e_2 belong to the same 2-factor TF_d, we have $\frac{i-j}{2} = \frac{m-n}{2} = d$ $(**)$.

From $(*)$ and $(**)$ we have $i = m$, which is contradictory to $i \neq m$. Therefore, all the colors on the edges in the 2-factor TF_d are different. □

Theorem 5. *Put the $\frac{N-1}{2}$ 2-factor together (which forms the complete graph K_N), the colors on the $N-1$ edges incident to a vertex $S_{i,i+1}$ are different, and these colors are also different from the color of the vertex $S_{i,i+1}$.*

Proof. Prove by contradiction. Suppose among the $N-1$ edges that are incident to the vertex $S_{i,i+1}$, there exist two edges that have the same color, and these two edges are $e_1 = (S_{i,i+1}, S_{j,j+1})$ and $e_2 = (S_{i,i+1}, S_{k,k+1})$. Because e_1 and e_2 have the same color, they correspond to the same vertex. Without loss of generality, let us assume $\frac{i+j}{2}$ is an even number. Then we have $\frac{i+j}{2} = \frac{i+k}{2} \Rightarrow j = k$, which is impossible since e_1 and e_2 are not the same. Therefore, the colors on the $N-1$ edges that are incident to the vertex $S_{i,i+1}$ are all different. □

In our example, Figure 4(4a) and Figure 4(4b) show the rainbow-coloring for the two 2-factors of K_5.

Step #5 - expanding the rainbow-colored 2-factors of K_N back to 4-factors of K_{2N}: Here we expand each 2-factor of K_N back to a 4-factor of the $2N$-vertex complete graph K_{2N}. For each edge $e = (S_{i,i+1}, S_{j,j+1})$ in a 2-factor of K_N, we expand it to a *monochromatic complete bipartite graph* $K_{2,2}$:

$$MCB_{i,j} = (V_i + V_j, E_{ij}), \tag{8}$$

where V_i and V_j are the vertex pairs $\{S_i, S_{i+1}\}$ and $\{S_j, S_{j+1}\}$ in the original $2N$-vertex graph G, and E_{ij} is the edge set of $K_{2,2}$. Additionally, we give all the 4 edges in E_{ij} the same color as that on the edge $e = (S_{i,i+1}, S_{j,j+1})$ in K_N. After this process, every 2-factor in K_N is expanded to a 4-factor in K_{2N}. Since the $\frac{N-1}{2}$ 2-factors of K_N are different, we have obtained $\frac{N-1}{2}$ different 4-factors of K_{2N}. We prove in Theorem 6 that the $\frac{N-1}{2}$ different 4-factors of K_{2N} together with the first PRM obtained in the step #2 form the N-colored $2N$-vertex complete graph K_{2N} that has the properties of (1) to (5).

In our example, Figure 4(5a) and Figure 4(5b) show the two 4-factors of K_{10} that are converted from the two 2-factors of K_5. These two 4-factors and the first PRM obtained in the step #2 (i.e., Figure 4(2)) together form the complete graph K_{10} with the properties of (1) to (5) (Figure 5(a)).

Theorem 6. *The $\frac{N-1}{2}$ different 4-factors of K_{2N} together with the first PRM obtained in the step #2 form the N-colored $2N$-vertex complete graph K_{2N} that has the properties of (1) to (5).*

Proof. For the reasons that 1) the $\frac{N-1}{2}$ 4-factors are obtained by expanding each edge of the $\frac{N-1}{2}$ 2-factors of K_N to a complete bipartite graph $K_{2,2}$ and 2) each pair of unconnected vertices in a $K_{2,2}$ will be connected by an edge in the first PRM obtained in the step #1, we can conclude that the $\frac{N-1}{2}$ 4-factors together

1: $(\lambda_0[0], \lambda_1[0]), (\lambda_1[1], \lambda_2[0]), (\lambda_2[1], \lambda_0[1])$
2: $(\lambda_0[0], \lambda_1[1]), (\lambda_1[0], \lambda_2[1]), (\lambda_2[0], \lambda_0[1])$
3: $(\lambda_0[1], \lambda_1[0]), (\lambda_1[1], \lambda_2[1]), (\lambda_2[0], \lambda_0[0])$
4: $(\lambda_0[1], \lambda_1[1]), (\lambda_1[0], \lambda_2[0]), (\lambda_2[1], \lambda_0[0])$

Figure 6: Dividing the edges of a $CMCB$ that contains 3 MCBs, i.e., $CMCB = \{\lambda_0, \lambda_1, \lambda_2\}$, into four groups such that all the edges in each group have no common vertex.

with the first PRM form a $2N$-vertex complete graph K_{2N} (i.e., the properties (1) and (3)).

For the reasons that 1) the coloring process of the 2-factors of K_N is a rainbow coloring process (i.e., Theorem 4) and 2) the color of each monochromatic complete bipartite graph (MCB) is taken from the corresponding edge of the 2-factor of K_N, we can conclude that all the N colors appear on K_{2N} (i.e., the property (2)).

For the reasons that 1) the colors on the $N-1$ edges incident to a vertex $S_{i,i+1}$ of K_N are different (i.e., Theorem 5), 2) these colors are also different from the color of the vertex $S_{i,i+1}$ (i.e., Theorem 5) and 3) the color of the vertex $S_{i,i+1}$ is the same as the color on the edge (S_i, S_{i+1}) of K_{2N} (i.e., the step #2), we can conclude that all the N colors appear on the $2N-1$ edges that are incident to the vertex S_i (which is also true for S_j)(i.e., the property (4)).

For the reasons that 1) the colors on the $N-1$ edges incident to a vertex $S_{i,i+1}$ of K_N are different (i.e., Theorem 5) and 2) each edge in K_N is expanded to two edges in K_{2N}, we can conclude that each of the $N-1$ colors appears twice on the edges that are incident to the vertex S_i (also true for S_{i+1}). Furthermore, for the reasons that 1) the color on the vertex $S_{i,i+1}$ is different from the previous $N-1$ colors (i.e., Theorem 5) and 2) the color of the vertex $S_{i,i+1}$ is the same as the color on the edge (S_i, S_{i+1}) of K_{2N} (i.e., the step #2), we can conclude that the color on $S_{i,i+1}$ appear once on the edges that are incident to the vertex S_i (also true for S_{i+1}). Therefore, the property (5) is satisfied. □

Step #6 - decomposing the 4-factors of K_{2N} into perfect rainbow match-

1: $(\lambda_0[0], \lambda_1[0]), (\lambda_1[1], \lambda_2[0]), (\lambda_2[1], \lambda_3[1]),$ $(\lambda_3[0], \lambda_4[0]), (\lambda_4[1], \lambda_5[1]), \ldots, (\lambda_{n-2}[0], \lambda_{n-1}[0]), (\lambda_{n-1}[1], \lambda_0[1])$
2: $(\lambda_0[0], \lambda_1[1]), (\lambda_1[0], \lambda_2[1]), (\lambda_2[0], \lambda_3[1]),$ $(\lambda_3[0], \lambda_4[1]), (\lambda_4[0], \lambda_5[1]), \ldots, (\lambda_{n-2}[0], \lambda_{n-1}[1]), (\lambda_{n-1}[0], \lambda_0[1])$
3: $(\lambda_0[1], \lambda_1[0]), (\lambda_1[1], \lambda_2[1]), (\lambda_2[0], \lambda_3[0]),$ $(\lambda_3[1], \lambda_4[1]), (\lambda_4[0], \lambda_5[0]), \ldots, (\lambda_{n-2}[1], \lambda_{n-1}[1]), (\lambda_{n-1}[0], \lambda_0[0])$
4: $(\lambda_0[1], \lambda_1[1]), (\lambda_1[0], \lambda_2[0]), (\lambda_2[1], \lambda_3[0]),$ $(\lambda_3[1], \lambda_4[0]), (\lambda_4[1], \lambda_5[0]), \ldots, (\lambda_{n-2}[1], \lambda_{n-1}[0]), (\lambda_{n-1}[1], \lambda_0[0])$

Figure 7: Dividing the edges of a $CMCB = \{\lambda_0, \cdots, \lambda_{n-1}, \lambda_0\}$, where n is the number of MCBs in the $CMCB$ and n is greater than 3, into four groups such that all the edges in each group have no common vertex.

ings in K_{2N}: Finally, we decompose each of the 4-factors of K_{2N} obtained in the step #5 into 4 different PRMs such that the properties (6) and (7) are satisfied.

In the previous step, each edge $(S_{i,i+1}, S_{j,j+1})$ in a 2-factor TF_d ($d \in [1, \frac{N-1}{2}]$) of K_N is expanded to a monochromatic complete bipartite graph $MCB_{i,j}$. Furthermore, recall that TF_d is either a Hamiltonian cycle of K_N (when $GCD(N,d) = 1$) or a set of $GCD(N,d)$ disjoint $\frac{N}{GCD(N,d)}$-cycles (when $GCD(N,d) \neq 1$), where $GCD(N,d)$ is the greatest common divisor of N and d. Therefore, the 4-factor FF_d of K_{2N}, which is expanded from the 2-factor TF_d of K_N, is a spanning graph of K_{2N} consisting of either one *chained monochromatic complete bipartite* graph (notated as $CMCB$) (when $GCD(N,d) = 1$) or a set of $GCD(N,d)$ disjoint $CMCB$s (when $GCD(N,d) \neq 1$).

Since a monochromatic complete bipartite graph $MCB_{i,j}$ connects two pairs of unconnected vertices $\{S_i, S_{i+1}\}$ and $\{S_j, S_{j+1}\}$, each $CMCB$ in the 4-factor FF_d ($d \in [1, \frac{N-1}{2}]$) of K_{2N} can be expressed as

$$< \lambda_0, \lambda_1, \cdots, \lambda_{n-1}, \lambda_0 >, \tag{9}$$

where $n = \frac{N}{GCD(N,d)}$ is the number of MCBs contained in the $CMCB$, and λ_p ($p \in [0, n-1]$) is the p-th unconnected vertex pair in the $CMCB$: $\{S_{2pd\%2N}, S_{(2pd+1)\%2N}\}$. For example, for the first 4-factor FF_1 of K_{10} shown in Figure 4(5a), we have $\lambda_0 = \{S_0, S_1\}$, $\lambda_1 = \{S_2, S_3\}$, $\lambda_2 = \{S_4, S_5\}$, $\lambda_3 = \{S_6, S_7\}$ and $\lambda_4 = \{S_8, S_9\}$. Meanwhile, for the second 4-factor FF_2 of K_{10} shown in Figure 4(5b), we have $\lambda_0 = \{S_0, S_1\}$, $\lambda_1 = \{S_4, S_5\}$, $\lambda_2 = \{S_8, S_9\}$, $\lambda_3 = \{S_2, S_3\}$ and $\lambda_4 = \{S_6, S_7\}$.

Given a $CMCB$ in a 4-factor expressed in formula (9), we divide the edges of the $CMCB$ into 4 groups as follows. For the 4 edges of each MCB, we put them

into 4 groups respectively such that edges in the same group share no common vertex. Figure 6 and Figure 7 show the way we divide the edges. Figure 6 shows the case that the $CMCB$ has 3 MCBs, and Figure 7 shows the case that the $CMCB$ has more than 3 MCBs. In these two figures, $\lambda_p[0]$ and $\lambda_p[1]$ are the first and the second vertex of the vertex pair λ_p ($p \in [0, 1]$) respectively.

If a 4-factor of K_{2N} contains one $CMCB$ (when $GCD(N, d) = 1$), the four edge groups obtained by using the dividing method shown in Figure 7 are the 4 different PRMs. If the 4-factor contains several disjoint $CMCB$s (when $GCD(N, d) \neq 1$), we put the i-th ($1 \leq i \leq 4$) edge group of each $CMCB$ into the same group to form a PRM. Therefore, each 4-factor of K_{2N} leads to 4 different PRMs, and the $\frac{N-1}{2}$ different 4-factors of K_{2N} produce $2N - 2$ different PRMs. Adding the first PRM obtained in the step #1, we now have $2N - 1$ different PRMs of K_{2N}. Each of these PRMs instructs the $2N$ CH sequences rendezvous in one of the $2N - 1$ hopping slots of a hopping period.

Since the edges in the same PRM share no common vertex and the colors of the K_{2N}'s MCBs are different, the property (6) is satisfied. Meanwhile, since each edge is assigned to only one PRM, the $2N - 1$ PRMs are different (i.e., the property (7) is satisfied).

In our 5-rendezvous channel network example (i.e., $N = 5$), using the dividing method in Figure 7, the first 4-factor of K_{10} (Figure 4(5a)) is decomposed into four different PRMs shown in Figure 4 (6a-1) to (6a-4), and the second 4-factor of K_{10} (Figure 4(5b)) is decomposed into another four different PRMs shown in Figure 4 (6b-1) to (6b-4). Based on the 9 PRMs (i.e., Figure 4(2), Figure 4 (6a-1) to (6a-4) and Figure 4 (6b-1) to (6b-4)), we construct the final $2N$ CH sequences shown in Figure 5(b).

Algorithm 5: Single-phase CH sequence construction

Data: N rendezvous channels $C_0, \cdots, C_{N-1}$, where N is an odd number;
Result: $2N$ CH sequence $S_0, \cdots, S_{2N-1}$, each of which has $2N-1$ slots.

$S_0, \cdots, S_{2N-1} \leftarrow \phi$;
for $i \leftarrow 0$ **to** $N-1$ **do**
  $S_{2i} \leftarrow S_{2i} \cup (0, C_i)$;
  $S_{2i+1} \leftarrow S_{2i+1} \cup (0, C_i)$;
for $d \leftarrow 1$ **to** $\frac{N-1}{2}$ **do**
  $sl \leftarrow 1 + 4(d-1)$;
  for $a \leftarrow 0$ **to** $GCD(N, d) - 1$ **do**
    $i \leftarrow 2a$;
    for $c \leftarrow 0$ **to** $\frac{N}{GCD(N,d)} - 1$ **do**
      if $\frac{i+(i+2d)\%2N}{2}$ *is an even number* **then**
        $u \leftarrow \frac{i+(i+2d)\%2N}{4}$;
      else
        $u \leftarrow \frac{\frac{i+(i+2d)\%2N+2N}{2}\%2N}{2}$;
      if $c == 0$ **then**
        AddSlot$(i, d, u, 0, 0, sl)$;
        AddSlot$(i, d, u, 0, 1, sl+1)$;
        AddSlot$(i, d, u, 1, 0, sl+2)$;
        AddSlot$(i, d, u, 1, 1, sl+3)$;
      else if $c == 1$ **then**
        AddSlot$(i, d, u, 1, 0, sl)$;
        AddSlot$(i, d, u, 0, 1, sl+1)$;
        AddSlot$(i, d, u, 1, 1, sl+2)$;
        AddSlot$(i, d, u, 0, 0, sl+3)$;
      else if $c == 2$ **then**
        AddSlot$(i, d, u, 1, 1, sl)$;
        AddSlot$(i, d, u, 0, 1, sl+1)$;
        AddSlot$(i, d, u, 0, 0, sl+2)$;
        AddSlot$(i, d, u, 1, 0, sl+3)$;
      else if c *is an odd number* **then**
        AddSlot$(i, d, u, 0, 0, sl)$;
        AddSlot$(i, d, u, 0, 1, sl+1)$;
        AddSlot$(i, d, u, 1, 1, sl+2)$;
        AddSlot$(i, d, u, 1, 0, sl+3)$;
      else
        AddSlot$(i, d, u, 1, 1, sl)$;
        AddSlot$(i, d, u, 0, 1, sl+1)$;
        AddSlot$(i, d, u, 0, 0, sl+2)$;
        AddSlot$(i, d, u, 1, 0, sl+3)$;
      $i \leftarrow (i + 2d)\%2N$;
return $S_0, \cdots, S_{2N-1}$;

Algorithm 6: Subfunction `AddSlot()` of Alg. 5

void AddSlot(i, d, u, a, b, sl) {
  $S_{i+a} \leftarrow S_{i+a} \cup (sl, C_u)$;
  $S_{(i+2d)\%2N+b} \leftarrow S_{(i+2d)\%2N+b} \cup (sl, C_u)$;
}

The complete algorithm

The complete single-phase CH sequence construction algorithm is given in Algorithm 5 with its subfunction `AddSlot()` shown in Algorithm 6. Given a DSA network with N rendezvous channels, the algorithm outputs $2N$ CH sequences, each with

$2N-1$ hopping slots, such that the following three conditions are satisfied. *First*, within the $2N-1$ hopping slots, every CH sequence meets with all the other $2N-1$ sequence each at a time in a hopping slot. *Second*, there are exactly two CH sequences hopping to the same rendezvous channel in a hopping slot. *Third*, in a CH sequence, each of the N rendezvous channels has the same probability to appear in the $2N-1$ hopping slots.

The algorithm essentially integrates the six intuitive steps described previously. Lines 2 to 4 of the algorithm schedule how the $2N$ CH sequences meet with each other in the slot-0, which is equivalent to forming the first PRM in step #1. Each iteration of the for-loop (lines 6 to 39) outputs rendezvous schedules for 4 hopping slots, which correspond to the 4 PRMs decomposed from a 4-factor of K_{2N}. Each iteration of the for loop (lines 8 to line 39) correspond to dividing the edges of a $CMCB$ into 4 groups (i.e., step #6). Lines 10 to 13 decides the channel to assigned, which correspond to the rainbow-coloring of 2-factors of K_N in step #4.

2.3.3 CH sequence execution

At the completion of constructing CH sequences by using either the two-phase CH sequence construction algorithm or the single-phase CH sequence construction algorithm, the newly joined node obtains a set of CH sequences, which are the same as those that any other nodes construct. Then the node synchronizes to the existing nodes using the global synchronization mechanism, and starts the channel hopping process described as follows. The node randomly selects a CH sequence to hop on. After hopping through all the slots, it performs the random CH sequence selection again and starts hopping on the newly chosen CH sequence. The node repeats this process while it is idle. The reason for the node to re-select a CH sequence after a hopping period is to make sure that any pair of nodes are able to rendezvous in different rendezvous channels. Since the selection of CH sequence is random, the requirement of full utilization of rendezvous channels

is satisfied. When a rendezvous channel's primary user appears, the nodes on that channel should yield using the channel, wait until a hopping slot, in which the rendezvous channel is available, is reached, and resume the hopping process.

2.4 ASYNC-ETCH

Our study of the communication rendezvous so far is based on the assumption that there exists a global synchronization mechanism to synchronize the hopping processes of the nodes. In this section, we investigate the design of CH based communication rendezvous without leveraging the synchronization mechanism. Without synchronization, a pair of nodes wishing to communicate with each other start channel hopping at a random time. Consequently, their CH sequences are most probably misaligned and SYNC-ETCH cannot guarantee channel overlap for rendezvous. We develop an asynchronous scheme, ASYNC-ETCH, to address the issue.

ASYNC-ETCH follows the similar steps: the CH sequence construction and CH sequence execution. ASYNC-ETCH constructs the CH sequences in a similar fashion as SeqR [51] but employs a novel enhancement: it constructs multiple CH sequences rather than only one as in SeqR. The arrangement of having multiple sequences brings two benefits. First, multiple sequences reduce the chance that two nodes select the same CH sequence. As we will show later, it takes less time for two nodes to rendezvous when they select different sequences. Second, with multiple sequences, participating nodes have more chances to rendezvous with each other within a hopping period. We show that a pair of nodes using ASYNC-ETCH that select two different CH sequences are guaranteed to rendezvous in N slots (where N is the number of rendezvous channels) within a hopping period no matter how the hopping processes of the pair of nodes are misaligned.

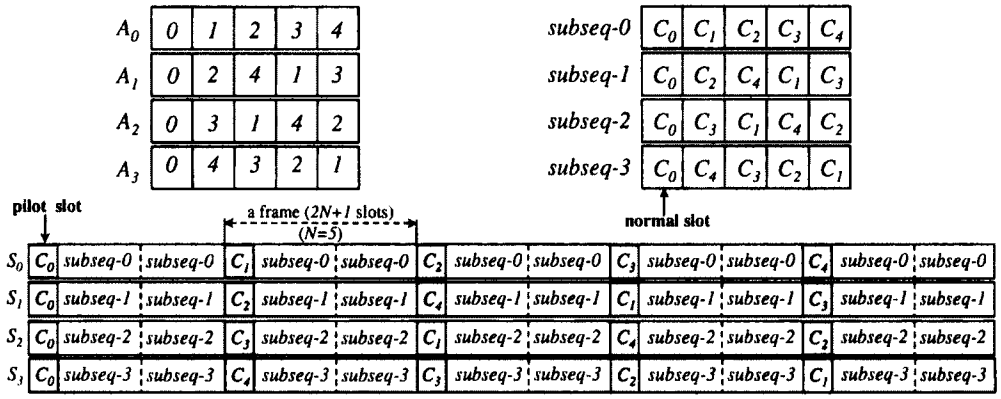


Figure 8: CH sequences of a DSA network with 5 rendezvous channels.

2.4.1 An overview and an example

In a DSA network with five (N) rendezvous channel, the nodes first construct a set of four ($N-1$) CH sequences, S_0, S_1, S_2 and S_3, as shown in Figure 8. As we can see from the lower part of the figure, each CH sequence consists of five (N) frames, each of which contains 11 ($2N+1$) slots: a pilot slot followed by two five-slot (N-slot) subsequences. The arrangement of the pilot slots is displayed in the the upper left part of the figure where pilot slot sequences A_0, A_1, A_2, A_3 are used in CH sequences S_0, S_1, S_2 and S_3, respectively. The arrangements for A_0 to A_3 are derived by the method of addition modulo the prime number five (N) with different addends from one to four respectively. The construction of the four subsequences (shown in the upper right part of the figure) also follows the channel assignment order determined in A_0 to A_3. As we will prove later, the above CH sequence construction guarantees that any pair of nodes (selecting two different sequences) rendezvous in N slots within a hopping period regardless how much channel hopping misalignment between the two nodes. Each ASYNC-ETCH CH sequence has 55 slots ($N * (2N+1)$).

After finishing the CH sequences construction, the nodes start the same CH hopping execution as in SYNC-ETCH: each of them randomly selects a CH sequence to start, and randomly reselects another one to continue after hopping on

Algorithm 7: Async. CH sequence Construction

Data: $\mathcal{C} = \{C_0, \cdots, C_{N-1}\}$: N rendezvous channels (N is prime).
Result: $S_0, \cdots, S_{N-2}$: $N-1$ final CH sequences.

1 **for** $i \leftarrow 0$ **to** $N-2$ **do**
2 | $A_i[0] \leftarrow 0$;
3 | **for** $j \leftarrow 1$ **to** $N-1$ **do**
4 | | $A_i[j] \leftarrow (A_i[0] + j(i+1))$ mod N;
5 **for** $i \leftarrow 0$ **to** $N-2$ **do**
6 | **for** $j \leftarrow 0$ **to** $N-1$ **do**
7 | | $subSeq_i[j] \leftarrow C_{A_i[j]}$;
8 **for** $i \leftarrow 0$ **to** $N-2$ **do**
9 | $k \leftarrow 0$;
10 | **for** $j \leftarrow 0$ **to** $2N^2+N-1$ **do**
11 | | **if** j mod $(2N+1) == 0$ **then**
12 | | | $S_i \leftarrow S_i \cup (j, subSeq_i[\frac{j}{2N+1}])$; *// pilot slot*
13 | | **else**
14 | | | $S_i \leftarrow S_i \cup (j, subSeq_i[k])$; *// normal slot*
15 | | | $k \leftarrow (k+1)$ mod N;
16 **return** $S_0, S_1, \cdots, S_{N-2}$;

the old one for a hopping period. By doing this, we ensure that any pair of nodes can rendezvous in different channels, which satisfies the requirement of full utilization of rendezvous channels. This arrangement also eliminates the unfairness that nodes selecting the same CH sequence have less chance to rendezvous than nodes selecting different CH sequences.

2.4.2 CH sequences construction

Algorithm 7 describes the construction of the $N-1$ CH sequences in ASYNC-ETCH. To ease our presentation, we assume the number of rendezvous channels, N, is a prime number. We hold the discussion of a general case (where N is not prime) till §2.4.4.

Given N rendezvous channels, ASYNC-ETCH first derives $N-1$ integer sequences A_0 through A_{N-2} (which will be used as indices for later channel assignment) by applying addition modulo the prime number N (lines 1 to 4). Note that all the integer sequences are derived with different addends. In lines 5 to 7,

the algorithm constructs $N-1$ CH sub-sequences, $subSeq_0$ to $subSeq_{N-2}$, whose channel indices are the same as the integer sequences A_0 through A_{N-2} respectively. Next, the algorithm constructs the CH sequence S_i $(0 \leq i \leq N-2)$ by concatenating five frames of S_i together (line 8 to 15). Each frame of S_i consists of a *pilot slot* followed by a pair of $subSeq_i$. Slots in $subSeq_i$ are referred as *normal slots*. The channels in S_i's pilot slots, combined together, are exactly channels appearing in $subSeq_i$ in the same order. From Algorithm 7, it is easy to see that ASYNC-ETCH fulfills the requirement of even use of the rendezvous channels.

2.4.3 Proof of rendezvous

In ASYNC-ETCH, the TTR between a pair of nodes is related to the fact that whether the two nodes select the same CH sequence or two different ones. Here we provide the theoretical analysis to determine the TTR performance in the above two situations. In particular, we prove that the two nodes have at least one overlapped CH slot within a hopping period in the former case, and they can rendezvous at least N times in the latter one.

Let us first rewrite the definition of *rotation closure property* from QCH [7] as follows.

Definition 1. *Given a CH sequence S with p slots and a non-negative integer d, $\mathcal{R}(S,d) = \{(i, \mathcal{R}(S,d)[i]) \mid \mathcal{R}(S,d)[i] = S[(i+d) \bmod p]\}$ is called a rotation of S with distance d.*

Definition 2. *A CH sequence S with p slots is said to have the rotation closure property with a degree of overlapping m if $\forall d \in [0, p-1]$, $|S \cap \mathcal{R}(S,d)| \geq m$.*

For instance, considering a CH sequence with three hopping slots, $S = \{(0, C_0), (1, C_0), (2, C_1)\}$, the two possible rotations are $\mathcal{R}(S,1) = \{(0, C_0), (1, C_1), (2, C_0)\}$ and $\mathcal{R}(S,2) = \{(0, C_1), (1, C_0), (2, C_0)\}$. It is obvious that S has the rotation closure property with a degree of overlapping 1.

Different from the prior work in SeqR [51], ASYNC-ETCH constructs multiple CH sequence rather than a single one. We provide the following definition to distinguish one CH sequence from another.

Definition 3. *Two CH sequences, S_0 and S_1, each with p slots, are said be different if $\forall d \in [0, p-1]$, $S_1 \neq \mathcal{R}(S_0, d)$.*

It is obvious that the $N-1$ CH sequences constructed by Algorithm 7 are different, since the subsequences, which are the building blocks of the CH sequences, are different.

We first analyze the case that two nodes select the same CH sequence.

Lemma 1. *For two nodes periodically hopping on a CH sequence that has the closure property with a degree of overlapping m, they can rendezvous in at least $\frac{m}{2}$ slots within a hopping period no matter how their hopping processes are misaligned.*

Proof. This lemma has been proved in QCH [7]. □

Theorem 7. *For two nodes that select the same CH sequence constructed by Algorithm 7, they can rendezvous in at least 1 slot within a hopping period no matter how their hopping processes are misaligned.*

Proof. We need to prove that for any CH sequence S_i $(0 \leq i \leq N-2)$ returned by Algorithm 7, S_i has the rotation closure property with a degree of overlapping 2, which combined with *Lemma* 1 can lead to this theorem. Specifically, we need to prove $\forall d \in [1, p-1]$, $\exists a \neq b \in [0, p-1]$ such that $S_i[a] = \mathcal{R}(S_i, d)[a]$ and $S_i[b] = \mathcal{R}(S_i, d)[b]$, where $p = 2N^2 + N$ is the number of slots of S_i.

If d mod $(2N+1) = 0$ (i.e., the 0-th slot of both $\mathcal{R}(S_i, d)$ and S_i are both pilot slots), then all $subSeq_i$ in both S_i and $\mathcal{R}(S_i, d)$ are aligned, there are $2N^2$ different overlappings.

If d mod $(2N+1) \neq 0$ (i.e., the 0-th slot in $\mathcal{R}(S_i, d)$ is a normal slot while the 0-th slot in S_i is a pilot slot), then we find the 2 overlappings as follows.

First, $\forall m, n \in [0, N-1]$ $(m \neq n)$, we have $S_i[m(2N+1)] \neq S_i[n(2N+1)]$ (since the 0-th slot in S_i is a pilot slot) $\Rightarrow \bigcup S_i[p(2N+1)] = \{C_0, \cdots, C_{N-1}\}$, where $p = 0, \cdots, N-1$, and $\mathcal{R}(S_i, d)[m(2N+1)] = \mathcal{R}(S_i, d)[n(2N+1)] \in \{C_0, \cdots, C_{N-1}\}$ (since the 0-th slot in $\mathcal{R}(S_i, d)$ is a normal slot). Then there must exist a $p \in [0, N-1]$ such that $S_i[p(2N+1)] = \mathcal{R}(S_i, d)[p(2N+1)]$.

Second, for $k = 2N + 1 - d \bmod (2N+1)$, the k-th slot in $\mathcal{R}(S_i, d)$ is a pilot slot while the k-th slot in S_i is a normal slot. Similar to the previous case, we can conclude that there exits an $p \in [0, N-1]$ such that $S_i[p(2N+1)+k] = \mathcal{R}(S_i, d)[p(2N+1)+k]$. □

To determine the rendezvous performance when two nodes select two different CH sequences, we first give the definition of integer sequences derived by the method of addition modulo a prime number with different addends, and prove its overlap property.

Definition 4. *Two integer sequences,* $A = \{a_0, \cdots, a_{N-1}\}$ *and* $B = \{b_0, \cdots, b_{N-1}\}$ *where* N *is a prime number, are said to be derived by the method of addition modulo the prime number* N *with different addends* m *and* n *if* $a_i = (a_0 + im) \bmod N$, $b_i = (b_0 + in) \bmod N$, *where* $0 \leq a_0, b_0 \leq N-1$, $1 \leq i \leq N-1$ *and* $1 \leq m \neq n \leq N-1$.

Lemma 2. *Given two integer sequences derived by the method of addition modulo a prime number with different addends* m *and* n, $A = \{a_0, \cdots, a_{N-1}\}$ *and* $B = \{b_0, \cdots, b_{N-1}\}$, *there must exist an integer* $t \in [0, \cdots, N-1]$ *such that* $a_t = b_t$.

Proof. Prove by contradiction. Suppose $\forall t \in [0, \cdots, N-1]$, $a_t \neq b_t$. Construct a integers sequence $C = \{c_0, \cdots, c_{N-1}\}$, where $c_i = a_i - b_i$ $(0 \leq t \leq N-1)$. It is easy to see that $\forall c_i, c_j \in C$ $(0 \leq i \neq j \leq N-1)$, $c_i \neq c_j$, otherwise we can get $a_0 - b_0 + i(m-n) \equiv a_0 - b_0 + j(m-n)$ $(\bmod N) \Rightarrow m - n$ is multiple times of N, which is impossible since $1 \leq m \neq n \leq N-1$. Because $a_t \neq b_t$ $\forall t \in$

$[0, \cdots, N-1]$, C contains N different integers that are in the range of $[1, N-1]$, which is a contradiction. □

Theorem 8. *For two nodes that select two different CH sequence constructed by Algorithm 7, there must be at least N overlapping slots within a hopping period between the two CH sequences no matter how their hopping processes are misaligned.*

Proof. Suppose S_i and S_j are two different CH sequences selected by the two nodes, we prove this theorem in the following two cases.

(1) The slot boundaries of S_i and S_j are aligned during the hopping processes of the two nodes. In this case, we have two further sub-cases as follows.

First, pilot slots in S_i overlap with pilot slots in S_j. In this case, all $subSeq_i$ in S_i exactly overlap with all $subSeq_j$ in S_j. Since integer sequences $\{A_i[0], \cdots, A_i[N-1]\}$ and $\{A_j[0], \cdots, A_j[N-1]\}$, which are the subscript sequences of $subSeq_i$ and $subSeq_j$ respectively, are derived by the method of addition modulo the prime number N with different addends, there exists one overlapping between a subsequence pair by *Lemma* 2. So there are $2N$ overlapping slots between S_i and S_j within a hopping period.

Second, pilot slots in S_i do not overlap with pilot slots in S_j. If the 0-th slot in S_i (a pilot slot) is aligned with the k-th $(0 \leq k \leq N-1)$ slot of the first $subSeq_j$ in a frame of S_j, then the first $subSeq_i$ in all the frames of S_i overlap with N contiguous normal slots in S_j. If the 0-th slot in S_i (a pilot slot) is aligned with the k-th $(0 \leq k \leq N-1)$ slot of the second $subSeq_j$ in a frame of S_j, then the first $subSeq_j$ in all the frames of S_j overlap with N contiguous normal slots in S_i. In either case, there exists at least one overlapping slot in each frame of both S_i and S_j because of *Lemma* 2 and the fact that the sequences of normal slots in S_i and S_j are developed by addition modulo prime the number N with different addends. So there are at least N overlapping slots between S_i and S_j within a hopping period.

(2) The slot boundaries of S_i and S_j are misaligned during the hopping processes of the two nodes. Suppose the first β $(0 < \beta < 1)$ portion of the 0-th slot in S_j overlaps with the l-th slot $(0 \leq l \leq 2N^2 + N)$ in S_i, then the rest $1 - \beta$ portion of the 0-th slot in S_j overlaps with the l'-th slot in S_i, where $l' = (l+1) \bmod (2N^2+N)$. Suppose the m-th slot in each frame of S_j is an overlapping slot if the boundaries of the 0-th slot in S_j and the l-th slot in S_i were aligned, and the n-th slot in each frame of S_j is an overlapping slot if the boundaries of the 0-th slot in S_j and the l'-th slot in S_i were aligned, then in each frame of S_j, S_j overlaps with S_i in the first β portion of the m-th slot and in the last $1 - \beta$ portion of the n-th slot. In other words, there is at least one overlapping slot in each frame of both S_i and S_j. So there are at least N overlapping slots between S_i and S_j within a hopping period. □

2.4.4 Additional discussion

Our previous analysis is based on the assumption that N is a prime number. To address the practical issue when N is not a prime number in a certain DSA network, we can make the following adjustment to easily remove the assumption. ASYNC-ETCH picks the smallest prime number that is greater than the number of rendezvous channels as the parameter N for Algorithm 7, and maps the excessive rendezvous channels down to the actual rendezvous channels. Figure 9 demonstrates an example of ASYNC-ETCH CH sequences construction in a DSA network with 4 rendezvous channels C_0 to C_3. ASYNC-ETCH first constructs 4 integer sequences A_0 to A_3 using addition modulo a prime number 5 with addends 1 to 4 respectively. Then it converts the integer sequences A_i to A'_i $(0 \leq i \leq 3)$ by replacing number 4 with number 0 in A_i $(0 \leq i \leq 3)$. Then the ASYNC-ETCH CH sub-sequences will be constructed according to integer sequences A'_i $(0 \leq i \leq 3)$. The drawback of this method is that some rendezvous channels are assigned more times to the CH sequences. Therefore, for DSA networks using ASYNC-ETCH, we recommend to assign a prime number of channels

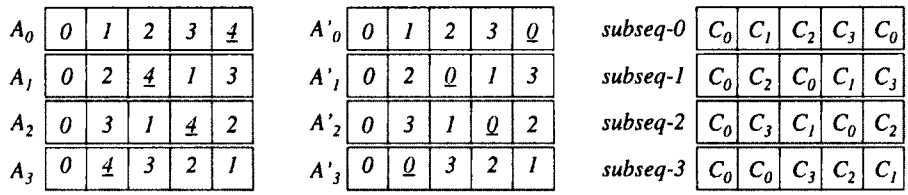


Figure 9: ASYNC-ETCH CH sequences construction in a DSA network with 4 rendezvous channels.

for control information exchange.

2.5 Comparisons

In this section, we theoretically compare ETCH with QCH [7] and SeqR [51], which are two existing CH based solutions for communication rendezvous in DSA networks.

In QCH, three versions of communication rendezvous protocols are designed. M-QCH and L-QCH are two synchronous versions that assume clocks are synchronized between nodes, and A-QCH is the asynchronous version that is used without such an assumption. The design goal of M-QCH is to minimize time-to-rendezvous between two CH sequences, while L-QCH's goal is to minimize the number of nodes that rendezvous in the same channel. SeqR is a DSA network communication rendezvous protocol without assuming global clock synchronization. SeqR does not have a synchronous version. We divide the comparisons into two group. In the first group, we compare SYNC-ETCH with M-QCH and L-QCH, all of which assume the existence of global clock synchronization. In the second group, we compare three asynchronous protocols: ASYNC-ETCH, A-QCH and SeqR.

We compare the two groups of communication rendezvous protocols on the three metrics introduced in §2.2.2: average rendezvous channel load, average TTR and rendezvous channels utilization ratio. Note that the choice of the CH sequence construction algorithm in the SYNC-ETCH protocol, i.e., the two-phase

Table 1: Comparisons between communication rendezvous protocols.

	Avg. Rend. channel load	Average TTR	Rend. channels utilization ratio
M-QCH	$\frac{2}{3}$	$\frac{3}{2}$	$\frac{1}{N}$
L-QCH	$\approx \frac{1}{\sqrt{2N-1}}$	$\frac{2N-1}{2}$	$\frac{1}{N}$
SYNC-ETCH	$\frac{1}{N}$	$\frac{2N-1}{2}$	1
A-QCH	$\frac{1}{2}$	$\geq \frac{9}{2}$	N/A
SeqR	$\frac{1}{N}$	$\frac{N^2+N}{2}$	N/A
ASYNC-ETCH	$\frac{1}{N}$	$\frac{2N^2+N}{N-1} \approx 2N$	N/A

algorithm or the single-phase algorithm, makes no difference on the protocol's theoretical performances on the three metrics, because we do not consider the impacts of the appearances of primary users in these theoretical comparisons. We will evaluate how the appearances of primary users have impacts on the performances of the SYNC-ETCH protocol using different CH construction algorithms later in §2.6.2.

Table 1 summarizes the comparison results, where N is the number of rendezvous channels of the DSA network. In the synchronous protocols group, we pick parameters for L-QCH such that it produces the same number of CH sequences as SYNC-ETCH for the purpose of fair comparison. SYNC-ETCH outperforms M-QCH and L-QCH on the metrics of average rendezvous channel load and rendezvous channels utilization ratio, because in every hopping slot it efficiently utilizes all rendezvous channels in establishing control channels, while there is only one channel can be used as control channel in each hopping slot with M-QCH and L-QCH. Thus theoretically, SYNC-ETCH experiences less traffic collisions and achieves higher throughput than QCH. For the metric of average TTR, M-QCH achieves the best theoretical performance. However, it has a very large average load on each rendezvous channel ($\frac{2}{3}$ of all the network nodes use the same rendezvous channel), which will cause a high probability of traffic collisions and further make the time-to-rendezvous performance of M-QCH worse

than its theoretical value in practice.

In the asynchronous protocols group, A-QCH has the worst performance in terms of average rendezvous channel load, because it only ensures two of the rendezvous channels can be used as control channels while both ASYNC-ETCH and SeqR utilize all the rendezvous channels in control channel establishment. Moreover, A-QCH cannot provide a bounded TTR. SeqR, which constructs only one CH sequence, can only guarantee one overlapping slot in a hopping period. So the average TTR for SeqR is half of the number of slots in the CH sequence (i.e., $\frac{N^2+N}{2}$). For ASYNC-ETCH's performance on the metric of average TTR, we make the following analysis: we proved in §2.4.3 that for the cases that when two nodes select the same CH sequence and when they select two different CH sequences, they are respectively guaranteed to meet in at least 1 slot and at least N slot within a hopping period. Since ASYNC-ETCH generates $N-1$ different CH sequences and the CH sequence selection is random, on average there are $\frac{1}{N-1} + \frac{(N-2)N}{N-1} = N-1$ guaranteed overlapping slots in a hopping period. So the average TTR for ASYNC-ETCH is $\frac{2N^2+N}{N-1} \approx 2N$.

2.6 Performance Evaluation

We evaluate ETCH's performance by simulation experiments. In §2.6.1, we compare ETCH with the existing CH based communication rendezvous protocols. In §2.6.2, we compare the two algorithms of SYNC-ETCH for CH sequence construction, i.e., the two-phase algorithm and the single-phase algorithm.

2.6.1 Comparing ETCH to the existing CH based communication rendezvous protocols

Methodology

We evaluate ETCH by comparing it to QCH and SeqR in the ns-2 simulator. We divide the evaluation into two portions based on the assumption about the existence of global clock synchronization. In §2.6.1, we compare the performances of SYNC-ETCH (using the two-phase algorithm for CH sequence construction), M-QCH and L-QCH. In §2.6.1, we compare the performances of ASYNC-ETCH with A-QCH and SeqR.

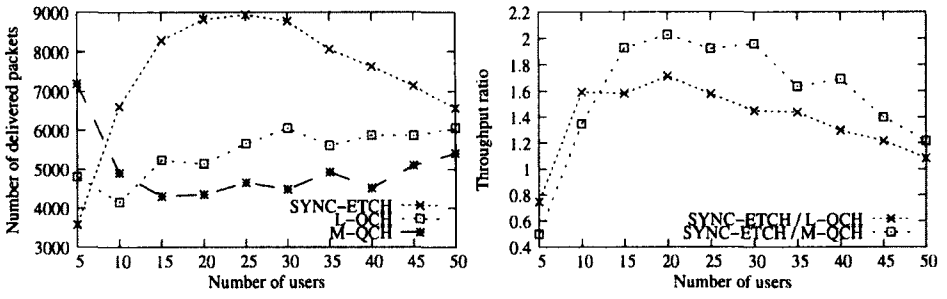
In the evaluation, we modify the ns-2 simulator to make it be able to perform multi-channel wireless communication simulations based on the Hyacinth project [56]. In our simulations, there are a varying number of nodes in a $500m \times 500m$ area, where each of the nodes is in all other nodes' communication ranges. The length of a hopping slot is set to 100 ms. We establish Constant Bit Rate (CBR) flows, where the packet size is set to 800 bytes and the packet rate is 125 packets/sec, from each node to all other nodes. These flows are started and stopped randomly during the simulation such that there is no more than one flow from the same node is activated simultaneously (because there is only one transceiver equipped with each node). Hyacinth's manual routing protocol is used in routing packets between the nodes. We disable the RTS/CTS function in the simulator, and rely on the retransmission mechanism to deal with packet collisions. In the simulations, the DSA network has 5 rendezvous channels (i.e., $N = 5$), each of which can possibly be used by the primary user. To simplify the simulation, we suppose all the secondary users are within the communication range of the primary user. The appearances of the primary user is simulated as follows. We first decide whether the primary user shows or not by flipping a coin. If the primary user appears, we randomly disable a rendezvous channel for a random period

of time. Otherwise all the rendezvous channels are made to be available to the nodes also for a random period of time. We repeat this process during the entire simulation.

Synchronous communication rendezvous protocols

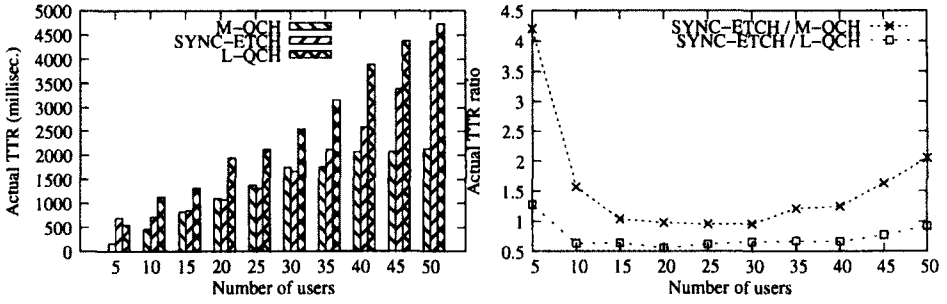
We conduct two simulation experiments to study the performances of the synchronous protocols on traffic throughput and actual time-to-rendezvous (TTR). In each experiment, we run the simulation for ten rounds with different number of secondary users (from 5 to 50 with a step length of 5) in each round.

Figure 10 shows the traffic throughput performances of the three synchronous protocols. Part (a) of this figure shows the actual throughput while part (b) illustrates the improvement ratio curves of SYNC-ETCH over L-QCH and M-QCH. SYNC-ETCH has a lower throughput than L-QCH and M-QCH when there are 5 secondary users in the network. This is because in CH sequences of L-QCH and M-QCH, rendezvous channels are randomly assigned to those non-frame-channel-slots, which may give a pair of nodes using L-QCH or M-QCH extra slots to rendezvous in other than the frame-channel-slot. And this is also because there are no or little collisions in this case. However, when the number of secondary users is equal or greater than 10, SYNC-ETCH achieves higher traffic throughput than L-QCH and M-QCH, especially when the nodes-channels ratio is in the range of 3 to 6 (i.e. when there are 15 to 30 nodes in the DSA network). In this case, traffic collision dominates the factors that influence the throughput performance. With both L-QCH and M-QCH, nodes are always compete for one rendezvous channel as control channel leaving all other rendezvous channels unused in a hopping frame, which causes a high probability of collisions when the nodes-channels ratio is bigger than 1. On the contrary, SYNC-ETCH schedules rendezvous among its CH sequences such that all the rendezvous channels can be utilized in every hopping slot. This approach greatly reduces traffic collisions



(a) Traffic throughput (b) Throughput ratio

Figure 10: Throughput performances of the synchronous protocols.



(a) Actual time-to-rendezvous (TTR) (b) Actual TTR ratio

Figure 11: TTR performances of the synchronous protocols.

and hence increases throughput. Furthermore, it can be also noticed in Fig 10 that the throughput performance of the three synchronous protocols converges as the nodes-channels ratio approaches 10. This is because collisions dominate traffics in each rendezvous channel with all the synchronous protocols. In this case, it is suggested to assign more rendezvous channels to accommodate such a high number of secondary users.

Figure 11 part (a) shows the TTR performances of the three synchronous protocols, and part (b) demonstrates the TTR ratios of SYNC-ETCH over L-QCH and M-QCH. The TTRs of the three protocols increase as the number of secondary users grows because of the increasing traffic collisions. Although M-QCH achieves the best TTR performance among the three as analyzed in §2.5, it does not get the theoretical TTR performance boost over SYNC-ETCH as analyzed in §2.5. Theoretically, M-QCH performs 3 times better than SYNC-ETCH in TTR, because it has an average TTR of 1.5 while SYNC-ETCH's value is 4.5. How-

ever, the simulation results shows that SYNC-ETCH's actual TTR is only 1.5 times of M-QCH's actual TTR on average. The reason of M-QCH's TTR performance degradation in the simulation experiment is because the nodes using M-QCH experience more severe traffic collisions that those using SYNC-ETCH.

From the above two simulations it can be seen that SYNC-ETCH achieves the best balance between traffic throughput and TTR among the three synchronous protocols.

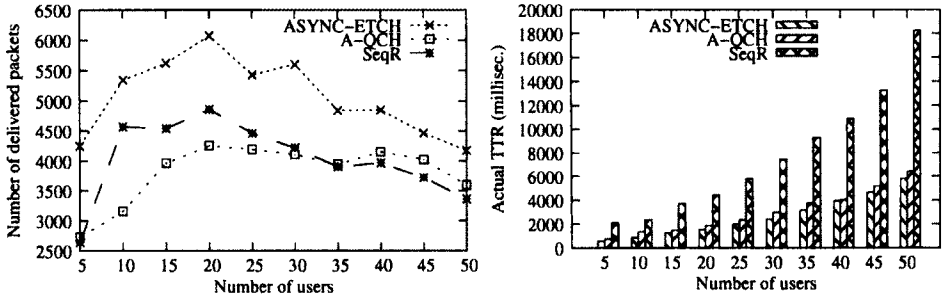
Asynchronous communication rendezvous protocols

In this subsection, we compare the throughput and the TTR performances between the three asynchronous protocols: ASYNC-ETCH, A-QCH and SeqR.

Figure 12 shows the performances of the three asynchronous protocols. In Figure 12 part (a), the traffic throughput performances are shown. ASYNC-ETCH performs constantly better than the other two protocols in this metric. This is because ASYNC-ETCH is able to utilize all the rendezvous channels as control channels while A-QCH uses only two of them. Meanwhile, ASYNC-ETCH improves on SeqR such that it achieves a shorter average TTR, which contributes to the throughput performance boost over SeqR. Figure 12 part (b) shows the actual TTR performances of the three protocols. It is not surprised that ASYNC-ETCH performance better than SeqR, because ASYNC-ETCH's average TTR is shorter than that of SeqR (see Table 1 for details). For A-QCH, we construct CH sequences such that they have an average TTR of 4.5, which is the best that A-QCH is able to achieve. Even so, ASYNC-ETCH still performs better than A-QCH.

2.6.2 Comparing the two algorithms in SYNC-ETCH

In the SYNC-ETCH protocol, we have proposed two algorithms for CH sequence construction. The two-phase algorithm can be applicable to DSA networks with an arbitrary number of rendezvous channels. However, it is unable to guarantee



(a) Traffic throughput (b) Actual time-to-rendezvous

Figure 12: Throughput and TTR of the asynchronous protocols.

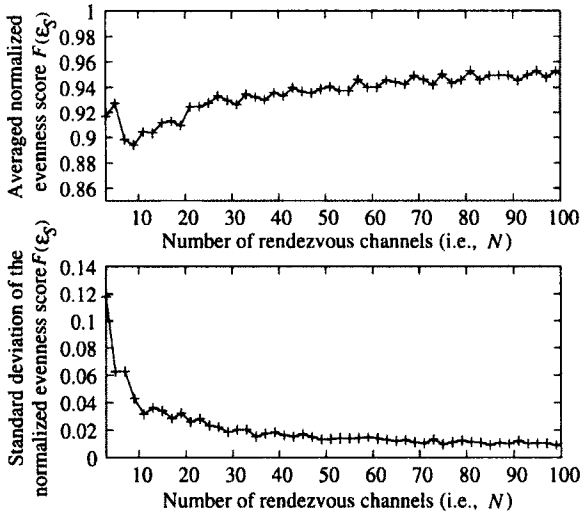


Figure 13: Channel appearance evenness score of the two-phase CH sequence construction algorithm. (The evenness score of the single-phase algorithm is always 1).

the even use of rendezvous channels requirement. The single-phase algorithm improves on its two-phase counterpart in that it guarantees, under the premise that N (i.e., the number of the rendezvous channels) is an odd number, all the rendezvous channels appear in each constructed CH sequence with the same probability.

To quantize how even the N rendezvous channels (i.e., $C_0, \cdots, C_{N-1}$) appear in a CH sequence S, we define the "evenness score" of S regarding rendezvous channel appearance probability as

$$\varepsilon_S = \sqrt{\frac{\sum_{i=0}^{N-1}(a_i - \frac{|S|}{N})^2}{N}},$$

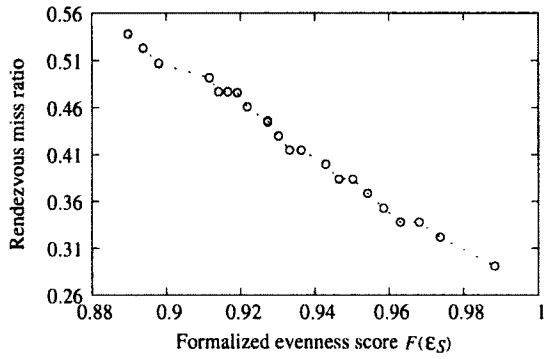


Figure 14: Rendezvous miss ratio vs. channel appearance evenness score.

where $|S|$ is the number of hopping slots of S, and a_i is the number of hopping slots of S in which channel C_i appears. We further convert ε_S into a normalized score $N(\varepsilon_S)$, which is the range of $[0, 1]$ and can be expresses as

$$N(\varepsilon_S) = 1 - \frac{\varepsilon_S - \varepsilon_{best}}{\varepsilon_{worst} - \varepsilon_{best}}.$$

In $N(\varepsilon_S)$, ε_{best} and ε_{worst} are the evenness scores of the best case and the worst case of fulfilling the even use of rendezvous channels requirement respectively. In the best case, each of the N rendezvous channels appears in S with the same the same probability, while in the worst case, a single channel appears in all the hopping slots of S. For instance, with the SYNC-ETCH protocol where there are $2N-1$ hopping slots in a CH sequence, the best case that a CH sequence S satisfies the even use of rendezvous channels requirement is that a rendezvous channels appears once in S while each of the remaining $N-1$ channels appears twice in S. The evenness score of the best case is calculated as $\varepsilon_{best} = \sqrt{\frac{(1-\frac{2N-1}{N})^2+(N-1)(2-\frac{2N-1}{N})^2}{N}}$. In the worst case, all the $2N-1$ slots is assigned with the same CH sequence. Accordingly, the evenness score of the worst case is calculated as $\varepsilon_{worst} = \sqrt{\frac{((2N-1)-\frac{2N-1}{N})^2+(N-1)(0-\frac{2N-1}{N})^2}{N}}$.

Low normalized evenness score of a CH sequence S indicates that S uses one or several rendezvous channels more than the remaining channels, which causes the nodes selecting S to have higher probability to experience communi-

cation outages if the primary users of those heavily relied channels show up. In SYNC-ETCH, every CH sequence constructed by the single-phase algorithm has a normalized evenness score of 1, which is the optimal case of fulfilling the even use of rendezvous channels requirement. To evaluate how well the two-phase algorithm satisfies this requirement, we calculate the average value and the corresponding standard deviation of the evenness scores of the $2N$ CH sequences constructed by the two-phase algorithm. Figure 13 shows the results of the cases where the value of N ranges from 3 to 99. The top graph of Figure 13 plots the average value of the evenness scores, and the bottom graph plots the corresponding standard deviations. We can see from the results that the two-phase algorithm still achieves an average normalized evenness score that is larger than 0.9 when N is greater than 10, and that the averaged score increases as N increases.

We further perform an experiment to evaluate how the normalized evenness scores of CH sequences affect the performances of the communication rendezvous protocol. In the experiment, we let a node A that is stick to a fixed CH sequence S_i rendezvous with another node B for $2N-1$ times, where the node B selects a different CH sequence S_j $(j \neq i)$ at each time. We disable γ $(0 < \gamma < 1)$ of the rendezvous channels that are used most frequently in S_i. The node A fails to rendezvous with the node B at a time if the overlapping channel between S_i and S_j is disable. We then calculate "rendezvous miss ratio" of the CH sequence S_i as the ratio between the number of times when a rendezvous attempt fails and the total number of rendezvous attempts (i.e., $2N-1$). Figure 14 (b) plots the relationship between the normalized evenness score and the rendezvous miss ratio of a CH sequence constructed by the two-phase algorithm S when $N = 33$ and $\gamma = 0.3$. Under the same settings, the rendezvous miss ratio of a CH sequence constructed by the single-phase algorithm is 0.27.

2.7 Conclusion

ETCH is a set of efficient channel hopping based communication rendezvous protocols for CR networks. ETCH protocols include SYNC-ETCH and ASYNC-ETCH. SYNC-ETCH, which assumes global clock synchronization, efficiently utilizes all the rendezvous channels in establishing control channels all the time. ASYNC-ETCH is able to make a pair of nodes rendezvous without being synchronized. Using a combination of theoretical analysis and simulations, we show that ETCH protocols perform better than the existing solutions for communication rendezvous in CR networks.

3 HoWiES: A Holistic Approach to ZigBee Assisted WiFi Energy Savings

Wireless communication service provides wireless connectivity to smartphone applications. However, it is also a major source of power consumption in smartphones. To provide efficient power management service in smartphones, we first studied how to reduce energy consumption for one of the most common wireless communication interfaces in smartphones: WiFi interface.

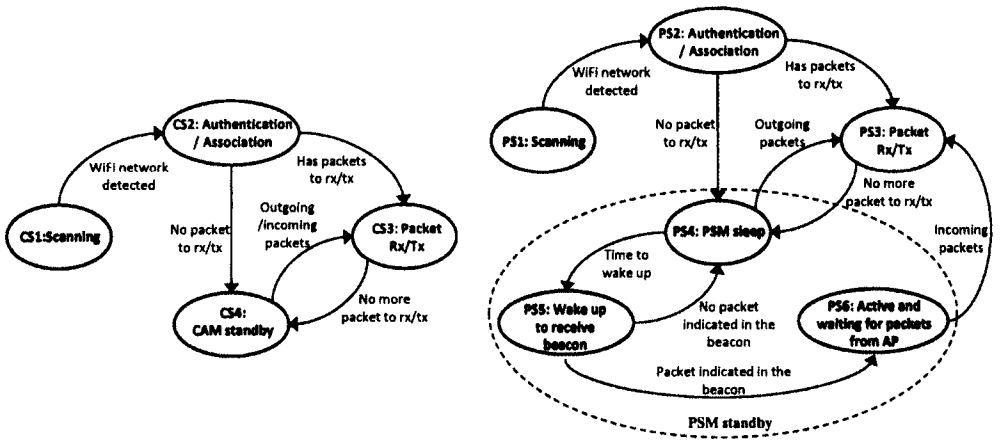
3.1 Background and related work

3.1.1 WiFi power management

The power management mode of WiFi stations[1] can be either CAM (Constantly Awake Mode) or PSM (Power Save Mode).

CAM stations keep their WiFi radio active all the time. Figure 15 (a) shows the operating states of CAM stations. After detecting and associating with a WiFi network, CAM stations switch their working states between "rx/tx" and "standby" (transitions between CS3 and CS4 in Figure 15 (a)): stations in the rx/tx state

[1]Mobile devices operating as stations in a infrastructure WiFi network as specified in the IEEE 802.11 standards.



(a) (b)

Figure 15: Operating state diagrams of CAM (a) and PSM (b) stations. Shaded states have room for energy savings.

are actively receiving and transmitting packets, while standby stations overhear all the packets in the air. Since the WiFi radio is active all the time, batteries in CAM stations drain at a rapid speed.

To save energy wasted in the CAM standby state, 802.11 power save mode is introduced [57]. Figure 15 (b) depicts the operating states of PSM stations. Similar to CAM stations, PSM stations also switch their working states between "rx/tx" and "standby" during operations. The difference is that PSM stations do not always keep their WiFi radios active in the standby state. Instead, PSM stations make their WiFi radios sleep (state PS4 in Figure 15 (b)) most of the time during standby. In the sleep state, WiFi radios consumes very low power but is not able to receive to transmit. PSM stations in sleep state switch to "rx/tx" state whenever they have outgoing packets (transition PS4 to PS3). To receive incoming packets, a sleeping PSM station needs to periodically switch its WiFi radio to active (transition PS4 to PS5, usually right before each beacon arrives) to receive its AP's beacons, through which the AP advertises buffered packets for its sleeping clients. If there is no packet indicated in the beacon for the PSM station, the station simply goes back to the PSM sleep state (transition PS5 to PS4). Otherwise, the station stays active and waits for its incoming packets from the AP (transition

Table 2: System power consumption in WiFi scanning state

	with WiFi scanning	with WiFi off	scanning/overall pert.
Galaxy S2	766 mW	265 mW	65.4%
T400	14498 mW	12732 mW	12.2%

PS5 to PS6). Then the station further switches to the "rx/tx" state on receiving the first incoming packet from its AP (transition PS6 to PS3). Upon completion of the receptions/transmissions, depending on detailed implementation, the station goes back to the sleep state either immediately or after a fixed amount of time without incoming or outgoing packets (transition PS3 to PS4). The default power management mode (i.e., CAM or PSM) of a station depends on the implementation of the WiFi driver. For example, ath5k and ath9k (i.e., the official Linux WiFi drivers for 802.11g and 802.11n Atheros chipset based stations respectively) [58,59] configure CAM as the default power management mode. Users can use the iw [60] utility to switch the power management mode between CAM and PSM. Drivers for Broadcom wireless chipsets being widely used in smartphones (e.g., BCM4329 chipset and BCM4330 chipset) configure PSM as the default power management mode.

3.1.2 WiFi energy saving opportunities

We observe that there are multiple significant energy saving opportunities for WiFi stations (i.e., mobile devices operating as stations in a infrastructure WiFi network as specified in the IEEE 802.11 standards.) in several of their working states, which are detailed as follows.

Opportunity 1 - scanning state: The first significant WiFi energy opportunity lies in the scanning state. Stations in scanning state constantly iterate through all the channels to search available WiFi networks. We have measured the system power consumption of two mobile platforms, a Samsung Galaxy S2 smartphone

Table 3: System power consumption in WiFi standby state

	with WiFi standby	with WiFi off	standby/overall pert.
Galaxy S2	298 mW	265 mW	11.1%
T400	14078 mW	12732 mW	9.6%

and a Lenovo T400 laptop, in the WiFi radio scanning state. From the measurement results (Table 2), we can see that about 65% and 12% of the system power consumption are spent in WiFi scanning for the Galaxy S2 smartphone and the Lenovo T400 laptop respectively. Moreover, recent research shows that people spend only half of their daily life in areas with WiFi signal coverages [61], which means their WiFi devices would spend about 12 hours a day in the high-power scanning state if they do not turn off WiFi radio when they are outside of WiFi coverages. Therefore, we are motivated to find an energy efficient way for mobile devices to discover WiFi networks instead of using power-hungry WiFi radios.

Opportunity 2 - standby state: The power management mode of WiFi stations can be either CAM (Constantly Awake Mode) or PSM (Power Save Mode). The difference between these two modes lies in when WiFi stations are in standby: a CAM station keeps its WiFi radio on all the time; a PSM station puts its WiFi radio into sleep (i.e., stay in a low-power state) for most of the time when there is no traffic, and periodically wakes up the radio to receive and check AP beacons, through which the AP informs the PSM stations about their packets buffered at the AP.

Table 3 presents the measurement results of the standby state power consumption of a Galaxy S2 smartphone and a T400 laptop, which are by default configured as PSM and CAM stations respectively by the device drivers. The Galaxy S2 smartphone consumes 33 mW more power, which accounts for about 11% of the overall system power, in the WiFi standby state than when the WiFi radio is turned off. This power overhead mainly comes from the periodic wakeup to check beacons, because when we increased the smartphone's wakeup inter-

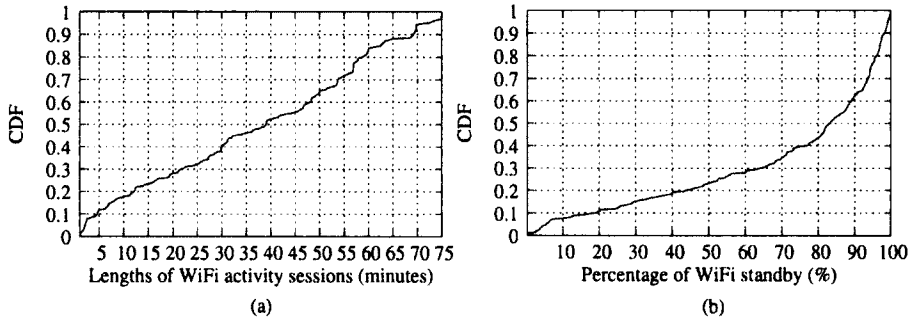


Figure 16: The experiment on WiFi standby time: (a) is the CDF of WiFi connection lengths. (b) is the CDF of the WiFi standby percentages in the corresponding WiFi connection sessions.

val, the power overhead decreased accordingly. The T400 laptop also consumes about 10 percent of its system power in the standby state. Recent works [10, 11] show that smartphone users usually leave their phones idle for most of the time, which makes the standby power consumption of WiFi radios even salient regarding saving energies for mobile devices. Ideally, WiFi radios should sleep without periodic wakeup or be completely turned off as long as there is no WiFi activities. Meanwhile, it must be possible to wake up the WiFi radios if there are incoming packets for them.

To study how much time stations spend in the standby state during WiFi connection sessions, we developed and deployed a WiFi activity recorder in our office building and in the college's library, both of which are heavy WiFi usage spots. The recorder sniffed all the WiFi packets and recorded their MAC addresses, packet types (e.g., data, management or control), packet sizes, data rates, received signal strength (RSS) and the packet reception times. To process the data, we first identified WiFi stations based on MAC addresses and packet types, and filtered out those stations whose packets have low RSS values, as the recorder may miss some of their packets because of low SNR. Then we analyzed the WiFi packets of the remaining stations to study how much time they were idle during WiFi connection sessions. Based on the 15 hours of WiFi activity data collected in 5 days, we identified 151 unique stations in 218 WiFi connection sessions. Figure 16 (a)

Table 4: Power consumption of CC2420 and BCM4330.

	CC2420 (ZigBee)	BCM4330 (WiFi)	ZigBee/WiFi ratio
Rx/Tx	56 mW	435 mW	0.129
Idle/Standby	1.2 mW	33 mW	0.036

plots the CDF of WiFi connection lengths, and (b) plots the CDF of the standby percentages in the corresponding WiFi connection sessions. If the reception time difference between two consecutive packets from the same station is larger than 5 seconds, we marked the interval between the two reception times as a WiFi standby duration of that station. Finally we concluded that in our measurement, over 70% of stations spent more than 60% of their time in standby during their WiFi sessions.

The notable standby power overhead (about 10%) and the large proportion of WiFi standby time over the entire WiFi session motivate us to design an energy-efficient way for WiFi standby. Ideally, WiFi radios should sleep without periodic wakeup or be completely off as long as there is no WiFi activities. Meanwhile, it must be still possible to wake up the WiFi radios if there are incoming packets for the WiFi radios.

Opportunity 3 - energy waste due to wakeup contention: When multiple PSM stations working at the same channel and associated either with the same AP [12] or with multiple co-located APs [13], are waken up to receive buffered packets at the same time, the contention between these stations will make them stay awake but without performing any communication tasks, which further causes about up to 4 times more energy consumption. Motivated by these research results, we want our approach to wake up standby WiFi radios to avoid these energy-expensive wakeup contentions.

3.1.3 ZigBee radio assisted WiFi energy savings

Compared with WiFi radios, ZigBee radios are more power efficient. Table 4 lists the power consumptions we measured of ZigBee radio CC2420 and WiFi radio BCM4330 in different operating modes. Since ZigBee is able to work at the same frequency band as WiFi while consumes significantly less energy, it would provide great assistance in saving WiFi energy for mobile devices if we could make ZigBee radios communicate with WiFi radios. Esense [62] is the first effort to enable communications between a WiFi radio and a ZigBee radio. The idea is using ZigBee radio to continuously sample the background energy in the air. Once there is a WiFi packet being transmitted, the sampling ZigBee radio will generate several consecutive samples whose energy readings are above a certain threshold, which we call *positive samples*. Esense studies how the number of consecutive positive samples (denoted as $\#^{+}_{consec}$) distributes when sampling WiFi packets replayed from several public WiFi traces. Esense proposes that each of those rarely occurring $\#^{+}_{consec}$ when sampling the public WiFi traces can be used to convey a certain message from WiFi to ZigBee. The experimental results of Esense show that it is able to deliver up to 100 different messages from WiFi to ZigBee.

The message capacity achieved by Esense is far from enough for being applicable to WiFi energy savings in mobile devices, since there could be up to 2007 stations associated with an AP [57]. Therefore, we are motivated to study how to extend the WiFi-ZigBee message capacity by using combinations of different $\#^{+}_{consec}$ to represent a message. Based on our new WiFi-ZigBee message delivery scheme, we design and implement three protocols that exploit the three opportunities to save WiFi energies for mobile devices.

3.1.4 Related work

Energy saving in WiFi scanning. To save the energy spent in scanning WiFi networks, several projects have considered, without turning on WiFi radios, predicting WiFi networks availability by using different context information [61], tracking and learning user movements [63], or collecting information about bluetooth devices and cell towers [64]. Turducken [65] proposes a heterogeneous devices architecture where a WiFi detector is used to detect whether WiFi signals are present. Similar to our solution, ZiFi [66] discovers WiFi networks with the assistance of ZigBee radios. The idea of ZiFi is using ZigBee to detect WiFi beacon patterns, which indicate the existence of WiFi networks. HoWiES takes a different approach: we enable APs to advertise themselves by broadcasting messages that are understandable by ZigBee radios. Thus, an advantage of our solution is that with HoWiES, mobile devices are able to selectively wake up and associate to the APs.

Energy saving in WiFi standby. To save the energy spent in WiFi standby, researchers have proposed to turn off WiFi radios when they are idle, and wake them up through a low-power non-WiFi channel when there are incoming WiFi activities. Wake-on-wireless [67] establishes the low-power channel by attaching a additional device to both APs and WiFi clients. Cell2Notify [68] considers using cellular channel to wakeup WiFi radios for VOIP calls. In our system, we establish the low-power channel directly between APs and devices' ZigBee radios through which APs can wake up standby devices selectively.

Energy saving in WiFi wakeup. Recent works have shown and addressed the energy waste problems caused by wakeup contentions between WiFi clients that belong to the same AP [12] or multiple interfering APs [13]. In our system, our solution naturally solves the problem of wakeup contentions between clients associated with the same AP by waking up WiFi clients one at a time. To alleviate

wakeup contention between clients associated with different APs, we coordinate APs such that there are not two interfering APs wake up their client at the same time.

3.2 System design

3.2.1 WiFi-ZigBee message delivery scheme

The high level idea. Let us assume the messages that WiFi radios can deliver to ZigBee radios correspond to different numbers. A WiFi radio encodes the number that it wants to convey to a ZigBee radio by sending a sequence of WiFi packets (called *WiFi-ZigBee message packets*), whose sizes are chosen from a group of predefined values, using a fixed transmission rate. These predefined packets sizes form the *alphabet* of our message delivery scheme. The ZigBee radio determines the size of each packet by sampling background energy, and obtains the number that the WiFi radio wants to convey by interpreting the combination of packet sizes.

Alphabet construction. The alphabet $\mathcal{A}$ is a set of b packet sizes: $\mathcal{A} = \{S_1, \cdots, S_b\}$, where $S_1 < \cdots < S_b$. In order to ensure that ZigBee radios can detect a WiFi-ZigBee message (abbreviated to "message" in later descriptions), we need to make message packets be distinguishable from normal WiFi packets. To this end, we carefully choose the predefined sizes for message packets and select the message packets transmission rate such that the air time of a message packet is longer than those of normal WiFi packets.

To study the air times of normal WiFi packets, we deployed WiFi sniffers in our office building and the university's library, both of which are heavy WiFi usage spots, and sniffed WiFi packets for three days. By looking at the sizes and the transmission rates of the sniffed packets, we observed that WiFi packets transmitted using low transmission rates were small in size (these packets were usually

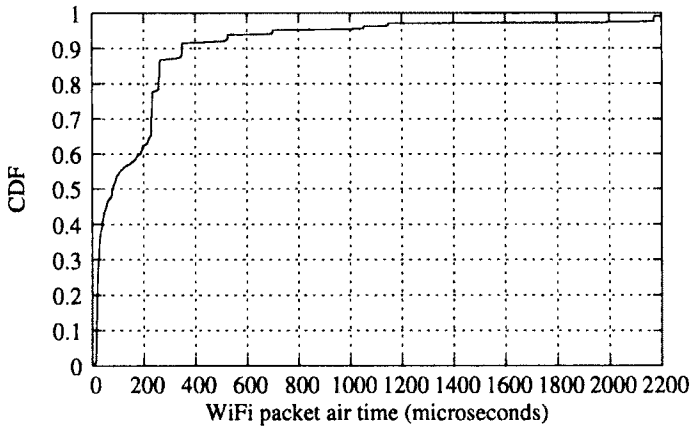


Figure 17: WiFi packets airtime distribution.

802.11 management/control frames like beacons and ACKs), and packets that were large in size were usually transmitted using high transmission rates (these packets were usually for massive data transmission like video streaming). Figure 17 shows the CDF of the sniffed packet airtime. From this figure we can observe that over 95% of all the sniffed packets had an air time less than 1 millisecond. Therefore, we ensure the air time of a message packet to be longer than those of normal WiFi packets by selecting large sizes for massage packets and sending them at the lowest transmission rate. Meanwhile, the difference between two adjacent predefined message packet sizes should be set appropriately to ensure ZigBee will not generate the same number of energy samples for message packets with different sizes. We will detail our choices of the predefined packet sizes for the alphabet later in §3.3.

WiFi-ZigBee message encoding: A WiFi radio encodes a WiFi-ZigBee message M by sending a sequence of l message packets, whose size are chosen from the alphabet $\mathcal{A}$, using the transmission rate R. Here we call l the *length* of the message. The value of the message is calculated as

$$v(M) = \sum_{i=1}^{i=l} (I_{p_i,\mathcal{A}} - 1) b^{i-1} \tag{3.1}$$

where b is the size of the alphabet $\mathcal{A}$, p_i represents the i-th of the l message

packets and $I_{p_i,\mathcal{A}}$ is the index of the packet p_i's size in the alphabet $\mathcal{A}$, for example, $I_{i,\mathcal{A}} = j$ if the size of packet p_i is S_j ($S_j \in \mathcal{A}, 1 \leq j \leq b$). Then the *capacity* of a message delivery scheme, which is the total amount of numbers that the scheme can encode, is b^l. Here R, l, b and $\mathcal{A}$ are fixed and shared between WiFi and ZigBee radios.

For instance, for a WiFi-ZigBee message delivery scheme where WiFi radios encode each message by transmitting 3 WiFi packets with sizes chosen from 100 and 200 bytes, the alphabet $\mathcal{A}$ is $\{100, 200\}$, the size of the alphabet b is 2 and the message length l is 3. The total number of messages that an WiFi radio can convey to a ZigBee radio is $2^3 = 8$ (i.e., the capacity of the scheme is 8). If a WiFi radio encodes a message by sending a sequence of 3 packets with 200B, 100B and 200B respectively, essentially it sends out 3 digits with values of 1, 0 and 1 in that order, and the message is interpreted as number 5 (i.e., $1 \times 2^0 + 0 \times 2^1 + 1 \times 2^2 = 5$).

Parameters selection: To ensure that ZigBee radios can distinguish WiFi-ZigBee message packets from background packets, we ensure the air time of message packets longer than the maximum packet air time of normal WiFi packets[2]. Suppose in a WiFi network the base transmission rate is R_w, then a WiFi-ZigBee message delivery scheme should choose its message packet transmission rate R and the smallest message packet size S_1 such that they satisfy $\frac{S_1}{R} > \frac{1500}{R_w}$, where 1500 is the Ethernet MTU [69]. Meanwhile, to guarantee ZigBee radios will not have the same energy sampling count for two message packets with different sizes, the difference between two adjacent message packet sizes should be at least $\frac{2R}{H}$, where H is the background energy sampling frequency of ZigBee radios.

WiFi-ZigBee message detection and decoding: Algorithm 8 presents the algorithm that ZigBee radios use to detect and decode WiFi-ZigBee messages. Zig-

[2]The maximum air time of ZigBee packets are smaller than that of WiFi packets.

Algorithm 8: WiFi-ZigBee message detection/decoding

Data: R, l, b, H, E and $\mathcal{A} = \{S_1, \cdots, S_b\}$.
Result: Report message value M once a message is detected.

```
1  PC, IC, i ← 0; state ← WAITING_MSG;
2  while ZigBee listening is enable do
3      Sample background energy, store the reading in e;
4      if (state == WAITING_MSG) then
5          if (e > E) /*on positive sample*/ then
6              PC ← 1; state ← PKT_IN_PROGRESS;
7      else if (state == WAITING_PKT) then
8          if (e > E) /*on positive sample*/ then
9              PC ← 1; state ← PKT_IN_PROGRESS;
10         else
11             IC++;
12             if (IC ≥ INTERVAL_TIME_OUT) then
13                 PC, IC, i ← 0; state ← WAITING_MSG;
14     else if (state == PKT_IN_PROGRESS) then
15         if (e > E) /*on positive sample*/ then
16             PC++;
17         else
18             if (PC ≥ HS_1/R) /*message packet detected*/ then
19                 i++;
20                 I_i ← j, if PC − ⌊HS_j/R⌋ < 2;
21                 if (i == l) /*message detected*/ then
22                     Report M = Σ_{i=1}^{i=l} (I_i − 1) b^{i−1};
23                     PC, IC, i ← 0; state ← WAITING_MSG;
24                 else
25                     IC ← 1; PC ← 0; state ← WAITING_PKT;
26             else
27                 if i == 0 /*no message packet has been detected*/ then
28                     PC, IC, i ← 0; state ← WAITING_MSG;
29                 else
30                     IC ← IC + PC;
31                     if (IC ≥ INTERVAL_TIME_OUT) then
32                         PC, IC, i ← 0; state ← WAITING_MSG;
33                     else
34                         PC ← 0; state ← WAITING_PKT;
```

Bee radios detect WiFi-ZigBee messages by continuously sampling background energy with a frequency H. If a sample's energy reading is greater than a threshold E, the sample is a "positive" sample, otherwise it is a "negative" sample. In the algorithm, the variable PC (positive sample counter) records the number of the

most recent consecutive positive energy readings that ZigBee radios have sampled, and the variable IC (message packet interval counter) records the time since the last message packet in terms of energy sample count. There are three working states in the algorithm. In the waiting message (WAITING_MSG) state (line 4-6), a ZigBee radio is waiting for a new WiFi-ZigBee message. Upon obtaining a positive sample it switches to the packet receiving (PKT_IN_PROGRESS) state (line 6). In the PKT_IN_PROGRESS state, the ZigBee radio keeps incrementing PC as it continuously gets positive samples (line 16). Upon receiving a negative sample, it decides whether the consecutive positive samples just observed come from a message packet or from a normal packet. If they come from a message packet (line 18-25), the ZigBee radio increments the message packet counter (line 19) and records the index of the packet's size in the alphabet (line 20). If all the message packets have been detected, it reports the message value based on the formula (3.1) (line 22), resets counters and switches back to the WAITING_MSG state (line 23). If there are message packets pending, it switches to the waiting message packet (WAITING_PKT) state (line 25). In the case that the consecutive positive samples come from a normal packet (line 27-34), the ZigBee radio switches back to the WAITING_MSG state directly if no message packet has been detected (line 28); otherwise, it counts the consecutive positive samples just observed into message packet interval (line 30). If the message packet interval is greater than a threshold, it switches back to the WAITING_MSG state (line 32). Otherwise, it goes to the WAITING_PKT state (line 34). In the WAITING_PKT state, the ZigBee radio keeps counting the message packet interval as they obtains negative samples (line 11), and ceases the decoding process if the interval is greater than the threshold (line 13). It goes to the PKT_IN_PROGRESS state once it obtains a positive sample (line 9).

Self-correcting message encoding/decoding. Without considering hidden terminals' effects, which is a case we will discuss at the end of this section, mes-

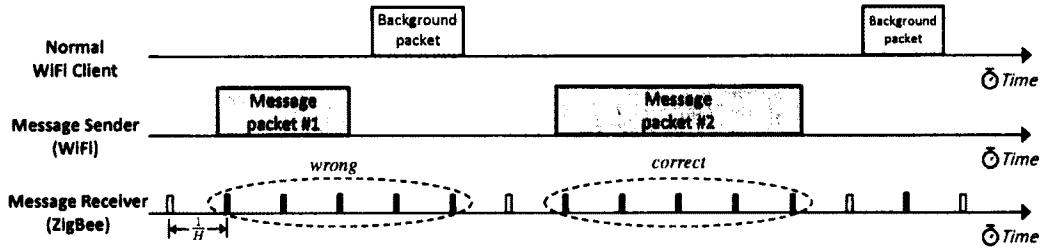


Figure 18: An example of background packet interference.

sage packets will not overlap with normal packets in time domain because of the 802.11 CSMA/CA scheme. However, since ZigBee radio cannot sample with an interval smaller than the IEEE 802.11 short interframe space (SIFS) (802.11 SIFS is 10 μs while the ZigBee standard [70] mandates that the energy reporting interval should be at least a symbol period (16 μs)), it is possible that a ZigBee radio obtains the same number of energy samples for two message packets with different sizes if there are two packets sent with an interval smaller than the ZigBee radio's sampling interval. Figure 18 shows an example: A WiFi radio sends out two WiFi-ZigBee message packets on which a ZigBee radio normally will generate 2-3 and 4-5 energy samples respectively. However, before the ZigBee radio could get the first sample after the first packet is transmitted, the channel is taken by another normal WiFi client, which transmits a packet causing the ZigBee radio to generate 2 positive samples on it. Then the positive samples of the first message packets is mistakenly counted as 5 instead of 3, which makes the ZigBee radio believe it has detected two message digits with the same value. We call this kind of problem *background (packet) interference*.

To address the above issue, we design a self-correcting message encoding/decoding algorithm, which extends the base encoding/decoding algorithm. With the self-correcting scheme, ZigBee radios can still extract the correct value of a message with high possibility even if background interferences exist. The fundamental observation supporting the self-correcting scheme is that when background interference happens, it only affects a minority amount of all the mes-

sage packets of a WiFi-ZigBee message. Thus, we can utilize the majority of correctly detected message packet sizes to help correcting those wrongly detected message packet sizes. With the self-correcting scheme, the alphabet $\mathcal{A}=\{S_1,\cdots,S_b\}$ is divided into p sub-alphabets as $\mathcal{A}_1=\{S_1,S_{p+1},S_{2p+1}\cdots\}$, $\mathcal{A}_2=\{S_2,S_{p+2},S_{2p+2},\cdots\}$, $\cdots$, $\mathcal{A}_p=\{S_p,S_{2p},\cdots S_b\}$. To encode a message, a WiFi radio uses packet sizes in one randomly chosen sub-alphabet. To decode a message, a ZigBee radio gets the sizes of all the message packets using Algorithm 8. If all the sizes are from the same sub-alphabet, the ZigBee radio can calculate the message value directly. Otherwise, it indicates that there were background interferences happened to the message packets. In this case, the ZigBee radio first identify the correct sub-alphabet (notated as $\mathcal{A}_c$) as the sub-alphabet to which the majority packet sizes belong. Then it converts each of those packet sizes that are not in $\mathcal{A}_c$ to the value in $\mathcal{A}_c$ that is immediately smaller than the current wrong size. This approach extends the difference between two adjacent predefined packet sizes in the alphabet by a factor of p, which makes it possible to tolerate multiple interfering background packets. Meanwhile, the capacity of the message delivery scheme is shrunk from b^l to $b(\frac{b}{p})^{l-1}$.

For instance, suppose there is a message delivery scheme where the alphabet is $\mathcal{A} = \{100, 200, 300, 400\}$ and message length is 3. An self-correcting scheme with two sub-alphabet (i.e., $p = 2$) allows WiFi radios to send a WiFi-ZigBee message by transmitting 3 packets with sizes chosen from one of the two sub-alphabets: $\mathcal{A}_1 = \{100, 300\}$ and $\mathcal{A}_2 = \{200, 400\}$. If a ZigBee radio detects that the sizes of the three message packets are 300B, 100B and 300B, which are from the same sub-alphabet, it can directly conclude that the message value is $1 \times 2^0 + 0 \times 2^1 + 1 \times 2^2 = 5$. If the packet sizes are 300B, 200B and 100B respectively, it indicates that $\mathcal{A}_1$ is the correct sub-alphabet as there are two packet sizes chosen from $\mathcal{A}_1$, and that the second packet (whose size is 200B) was affected by background interference. In this case, the ZigBee radio replaces the size 200B in

$\mathcal{A}_2$ with size 100B in $\mathcal{A}_1$, and reports the message value as $1\times2^0+0\times2^1+0\times2^2=1$.

3.2.2 HoWiES energy saving protocols

Based on the WiFi-ZigBee message delivery scheme, we design three HoWiES energy saving protocols that save energy consumed in WiFi scanning, standby and wakeup respectively. At the mobile device side, three components relate to HoWiES operations: The *WiFi component* performs the ordinary 802.11 operations. The *ZigBee component* acts as a receiver in the WiFi-ZigBee message delivery scheme. The *HoWiES manager* is a software component that connects the components of WiFi and ZigBee and performs all the HoWiES management operations. At the AP side, each AP has a pool of WiFi-ZigBee message numbers, each of which is assigned to deliver a certain piece of information from WiFi to ZigBee as specified in the following protocol descriptions.

HoWiES scanning and association. The HoWiES scanning and association protocol establishes a connection between APs and HoWiES-capable mobile devices. Figure 19 shows the protocol. With this protocol, mobile devices trying to search and join a HoWiES-enable WiFi network keep their WiFi radios off while using the ZigBee radio to detect WiFi network advertisement messages broadcast regularly by HoWiES-enabled APs (Op.1). Among all the WiFi-ZigBee message numbers, APs use a set of common numbers to advertise their networks (in the HoWiES scanning protocol) and to indicate buffered broadcast/multicast packets (in the HoWiES wakeup protocol). During the scanning process, a HoWiES client turn on its WiFi radio and associate to an AP based on the numbers encoded in the WiFi-ZigBee messages received. For example, a system operator can configure open APs to encode "1" in their network advertisement WiFi-ZigBee messages, and configure encrypted APs to encode "2". Then mobile devices can selectively turn on their WiFi radios based on whether the encountered networks is encrypted. Upon detecting an advertisement message (Op.2), the ZigBee com-

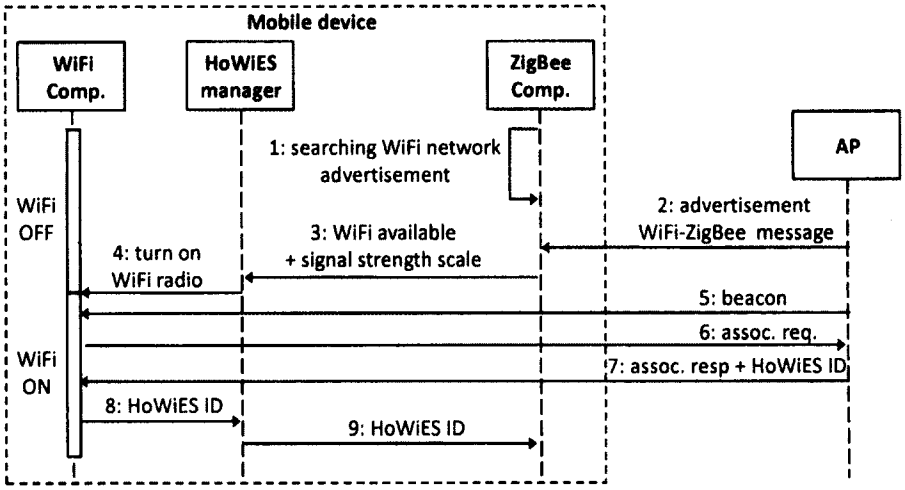


Figure 19: HoWiES scanning and association operations.

ponent notifies the HoWiES manager about the presence of a WiFi network and the scale of the WiFi signal strength calculated based on the energy samples of the message (Op.3). The HoWiES manager turns on the WiFi radio if the WiFi network meets the device's needs (Op.4). The WiFi radio sends an association request, indicating that the request issuer is HoWiES-capable, to the AP based on the information in the WiFi beacons (Op.5 and 6). If the association succeeds, the AP chooses a number from its message number pool to assign to the newly associated client as its HoWiES ID, and puts this ID in the association response (Op.7). Finally, the WiFi component extracts the ID from the association response and send it to the ZigBee radio via the HoWiES manager (Op.8-9).

HoWiES standby. This protocol puts mobile devices into HoWiES standby by turning off the WiFi radio and informing AP about the status change on the mobile devices. The upper half of Figure 20 shows the protocol. The HoWiES manager keeps monitoring the WiFi traffic on the mobile device (Op.1). On detecting that the WiFi radio has been idle for a certain amount of time, the HoWiES manager notifies the WiFi radio to go into HoWiES standby state (Op.2). Then the WiFi radio informs the AP that it will switch to the HoWiES standby state and then turns itself off for energy savings (Op.3). Right after notifying the WiFi component

to switch to HoWiES standby, the HoWiES manager turns on the ZigBee radio for WiFi-ZigBee message listening during standby (Op.2′). With this protocol, WiFi radios in HoWiES standby devices do not need to switch to active periodically to check beacons for buffered packets. Instead, they can just sleep all the time till the ZigBee radio detects wakeup messages sent from the AP.

In the above design, we let both CAM devices and PSM devices switch to HoWiES standby only when their HoWiES managers predict that the durations of inactivities are longer than a threshold. The purpose of this rule for CAM devices is obvious: we want to reduce the performance impact to CAM devices brought by standby wakeup delays as performance has higher priority in CAM devices. For PSM devices, this rule will greatly reduce the overheads generated by WiFi-ZigBee messages on both network throughput and AP performances. As we will see later, the message delivery scheme we have implemented has a negligible amount of overheads on network throughput and AP performances when the message sending frequency is less than 10 (i.e., 10 messages per second). However, when the sending frequency is larger than 10, the overheads increase linearly as the message sending frequency increases. Therefore, putting a PSM station into HoWiES standby only when the HoWiES manager predicts that the station will stay idle for a long duration will significantly reduce the overheads. The WiFi inactivity prediction could be achieved by combining statistical WiFi traffic history analysis [71] and user WiFi usage pattern learning. We leave the design of the inactivity prediction to the future work.

HoWiES wakeup. The bottom half of Figure 20 shows the HoWiES wakeup operations. During standby, the ZigBee component keeps listening for WiFi-ZigBee messages encoding the device's HoWiES ID (Op.4). Once the AP has buffered incoming packets for a HoWiES standby client, it wakes up the client by sending out a WiFi-ZigBee message that encodes the HoWiES ID assigned to the client in the association process (Op.5). If the buffered packets are broad-

cast/multicast packets, a common number, instead of the HoWiES ID, is encoded in the message. If there are multiple clients that have buffered packets, the AP wakes them up one by one in a FIFO manner. The ZigBee component informs the HoWiES manager about the buffered packets if it detects the number encoded by a WiFi-ZigBee message matches the device's HoWiES ID (Op.6). Then the HoWiES manager turns on the WiFi radio (Op.7), which in turn gets the buffered packets from the AP (Op. 8-9).

Since APs wake up its HoWiES standby clients one at a time, this approach naturally solves the wakeup contention problem causing by waking up multiple WiFi clients associated with the same AP. However, if multiple interfering APs (i.e., APs that can hear each other) wake up their own clients at the same time, the awake times of the clients due to the wakeup contentions could be extended by a factor of 5 [13]. To solve the problem, we let each AP exclusively occupies a repeated *wakeup period*, during which it can wake up its clients to get their buffered packets, such that wakeup periods of any two interfering APs do not overlap. An AP's wakeup period starts at the beginning of each of its beacon period (i.e., right after a beacon is sent out), and lasts a duration of T_{dur}. The value of T_{dur} is determined in the same way as the length of a fair share is determined in [13]. Interfering APs coordinate their beacon periods [13] to ensure their wakeup periods do not overlap with each other.

3.2.3 Discussions

Dealing with hidden terminals. In designing the self-correcting message encoding/decoding scheme, we assume that two WiFi packets will not overlap in time domain due to 802.11 CSMA/CA. However, if there are two hidden nodes transmitting without knowing each other, their packets could be concatenated in time domain at a certain place between them. In this case, the concatenated packet may have an air time equal to a WiFi-ZigBee message packet, causing a sampling

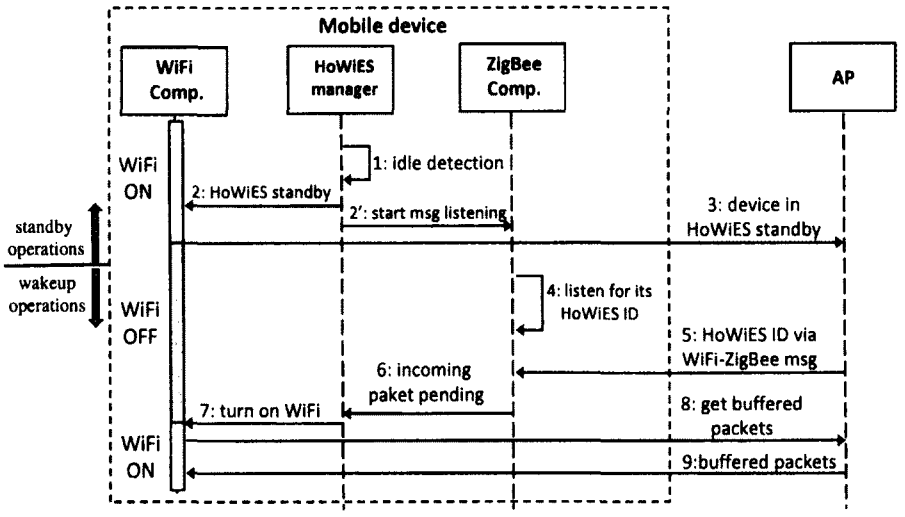


Figure 20: HoWiES standby and wakeup operations.

ZigBee radio to have wrong detections. Similar to the existing solutions dealing with the hidden terminal problems, we address this issue by using retransmissions: when an AP sends a message encoding a client's HoWiES-ID to wake up the client, it will keep sending the message with a certain interval until the client wakes up and fetches the buffered packets.

Variable message length. In our current design, all WiFi-ZigBee messages have the same length (i.e., use the same number of packets to encode different messages). A promising way to increase the efficiency of the message delivery scheme is to use less packets to encode those frequently used messages and more packets to encode those rarely used messages (which is an idea similar to Huffman coding). We leave this exciting improvement to our future work.

3.3 System implementation

We have implemented the HoWiES system with the devices shown in Figure 21. The system consists of two types of entities: HoWiES clients and HoWiES APs. HoWiES clients are implemented in two mobile platforms: a smartphone platform (Samsung Galaxy S2) and a laptop platform (Lenovo T400). We enable ZigBee

Table 5: OSes and WiFi drivers of implementation devices

Device	Operating System	Driver
Samsung Galaxy S2	Android 2.3 (Linux 2.6.35)	DHD
Lenovo T400	Ubuntu 10.04 (Linux 2.6.32)	ath9k
Dell Latitude D820	Ubuntu 10.04 (Linux 2.6.32)	madwifi
Wiligear WBD-500	OpenWrt 8.09 (Linux 2.6.26)	madwifi

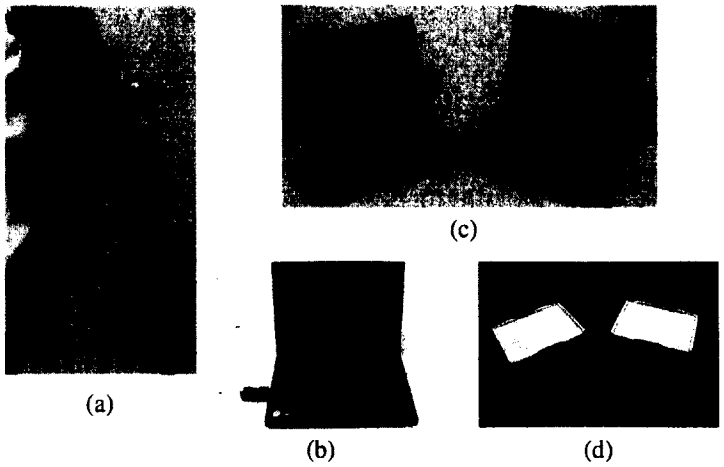


Figure 21: (a) HoWiES client implemented in a Samsung Galaxy S2 smartphone. (b) HoWiES client implemented in a Lenovo T400 laptop. (c) HoWiES APs implemented in Dell Latitude D620/D820 laptops. (d) HoWiES APs implemented in the Willigear WBD-500 integrated radio platform.

in both mobile platforms by integrating each of them with a TelosB mote that has a CC2420 ZigBee radio via USB interface. HoWiES APs are implemented in two AP platforms: a laptop platform (Dell Latitude D620/D820) and a standalone AP platform (Wiligear WBD-500 integrated radio platform). Table 5 lists the OS and the WiFi driver used in each device.

3.3.1 HoWiES client

A HoWiES client has three major components: the WiFi component (consisting of the WiFi radio and the WiFi driver), the ZigBee component (consisting of the CC2420 ZigBee radio and the message detection/decoding TinyOS module) and the HoWiES manager.

Background energy detection: The CC2420 ZigBee radio has an RSSI register that records the RSS averaged over 8 symbol periods. The TinyOS provides an interface for programs to read the value of the RSSI register. However, according to our experience, the native TinyOS interface needs around 500 μs to get an RSS reading from the register. To increase the RSS sampling rate (so as to have more packet sizes for the alphabet), we have managed to reduce the RSS reading interval to about 150 μs. In our implementation, we set the ZigBee RSS reading interval to 180 μs (i.e., $H = 5555$) for stable performances.

Message detection/decoding: The ZigBee component continuously detects and decodes all WiFi-ZigBee messages by running Algorithm 8, and notifies the HoWiES manager about the messages that are related to the hosting mobile device (e.g., WiFi network advertisements and the device's HoWiES ID).

Duty cycling ZigBee radio: According to our measurement, the power consumption that a TelosB mote has when it is sampling background energy is about 60 mW, which is higher than the standby WiFi power overheads in Galaxy S2 (33 mW). To solve this issue, we adopted a solution similar to [72], where the sensor is put to sleep periodically for energy savings. We have reduced the energy sampling power consumption of TelosB mote to 5 mW by duty cycling the ZigBee radio. In our implementation, a ZigBee radio samples background energy only during the wakeup period of the AP that its hosting device is associated with. To synchronize ZigBee radios with the corresponding APs' wakeup periods, we let APs broadcast the durations of their current wakeup periods (i.e., T_{dur}) via beacons. Then the HoWiES manager enables ZigBee energy sampling only in the first T_{dur} of time of the corresponding AP's beacon period (recall that each AP's wakeup period starts at the beginning of its beacon period). Before an AP has to adjust its beacon period (because of topology changes of interfering APs), it wakes up all its HoWiES standby clients to let them be able to re-synchronize to its new wakeup period.

The HoWiES manager: The HoWiES manager is implemented as a Linux kernel module in the mobile device's OS. It is responsible for turning on/off WiFi radios as specified in the protocols, controlling background energy sensing in ZigBee radio and relaying information between the WiFi and the ZigBee components. The HoWiES manager communicate with the ZigBee component via USB serial connection.

3.3.2 HoWiES AP

WiFi-ZigBee message parameters selection: In our implementation, HoWiES APs send out a WiFi-ZigBee message by transmitting 3 packets (i.e., $l = 3$) with a transmission rate of 1 Mb/s (i.e., $R = 1$ Mb/s). We experimentally quantified how stable the CC2420 radio generates energy samples in sampling packets with a fixed length. We found that the CC2420 radio we used can produces 4 different numbers of energy samples for the same WiFi packet size. Therefore, to ensure ZigBee will not generate the same number of energy samples for two message packets with different sizes, we set the difference between two adjacent packet sizes in the alphabet to 90 bytes (i.e. $\frac{4R}{H}$), which gives us 14 packet sizes for the alphabet: $\mathcal{A} = \{300, 390, \cdots, 1470\}$. Thus, the smallest air time for the message packet is 2.4 millisecond, which is larger than the air times of all the sniffed WiFi packets obtained in our experiment described in the "Alphabet construction" subsection.

WiFi-ZigBee message packets transmission: HoWiES APs transmit message packets using a user space packet sending program implemented with the libpcap library. The user space program and the WiFi driver located in kernel space are connected by using the Linux usermode-helper API.

Table 6: Reliability and accuracy of the implemented WiFi-ZigBee message delivery scheme in the uncontrolled experiment.

Reliability	**Accuracy**	
Total msg detected/sent	Correct msg/detected (w/o self-correction)	Correct msg/detected (w/ self-correction)
19,904/20,000	19,223/19,904	19,737/19,904
99.5%	96.6%	99.2%

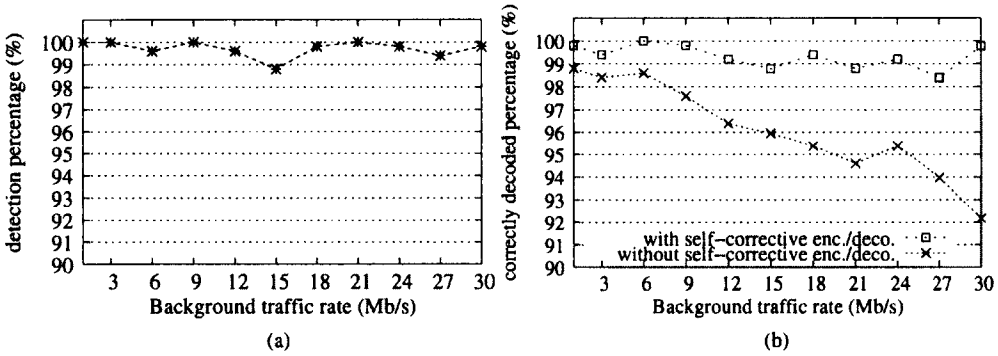


Figure 22: Reliability and accuracy of the implemented WiFi-ZigBee message delivery scheme in the controlled experiment. (a) shows the reliability performance: the message detection percentage vs. different background traffic bandwidth. (b) shows the accuracy performance: the correctly decoded message percentage vs. different background traffic bandwidth.

3.4 System evaluation

3.4.1 WiFi-ZigBee message delivery

Reliability and accuracy. The message delivery scheme needs to be *reliable*, which means HoWiES clients should reliably detect WiFi-ZigBee messages sent by HoWiES AP without firing any false alarms (i.e., reporting messages when there is none). Meanwhile, the message delivery scheme needs to be *accurate*, which means HoWiES clients should be able to correctly decode the detected messages.

We have performed an uncontrolled experiment to evaluate the reliability and the accuracy performances of the implemented message delivery scheme in real

WiFi environments. We deployed a HoWiES AP and client pair in the university's library, and performed the experiment from 8 PM to 10 PM, a time section during which the library are full of students surfing web and watching online videos, in several days. In the experiment, the AP sent different numbers to the client in different rounds. In each round, the HoWiES AP randomly chose a number from 1 to 2744, encoded the number into a WiFi-ZigBee message and transmitted the message for 100 times with an interval of 100 ms. The chosen number is recorded such that we can use it as ground truth when deciding if the client has correctly decoded the messages. The HoWiES client detected and decoded the messages using the base message encoding/decoding algorithm (i.e., without using the self-correcting scheme), and output the results to a data file for analysis. We ran the experiment for 200 rounds. Table 6 shows the results. For the total 20,000 WiFi-ZigBee messages, 99.5% of them were detected by the HoWiES client. Within all the detected messages, the HoWiES client correctly decoded 96.6% of them. We then examined all the wrongly decoded messages as follows. We marked an wrongly decoded message as correctable using the self-correcting scheme with 2 sub-alphabets (i.e., $p = 2$), if the following conditions are satisfied. First, there is only one message packet whose size is wrongly detected (since we use $l = 3$, one is the maximum minority number). Second, the wrong size's index in the alphabet is greater than the actual size's index in the alphabet by 1 (if using $p = 3$, this value is 2). We found that after using the self-correcting algorithm, the accuracy of the message decoding increased to 99.2%. We further examined what caused the rest uncorrectable messages. There are two reasons. The first reason is that some messages have more than one message packet whose size is wrongly detected. The second reasons is that although there is only one wrong message packet size, the energy samples count for that packet is less than the expected value. This might be because of the imperfection of CC2420 hardware implementation of energy detection.

We also conducted a controlled experiment to study how the message delivery reliability and accuracy performances respond to the changes of background traffic. In this experiment, we produced background traffic by establishing a direct iperf UDP connection between two 802.11g WiFi nodes (UDP packet size was 1500 bytes). We varied the connection bandwidth between the two nodes and observed how our message delivery scheme responded to that. We have tested background traffic bandwidth from 1 Mb/s to the *saturated* bandwidth (30 Mb/s) with a step length of 3 Mb/s. Similar to the uncontrolled experiment, the HoWiES AP transmitted messages encoding a randomly selected number, without using the self-correcting algorithm, for 100 times in each round. With each background traffic bandwidth, we performed the test for 100 rounds. Figure 22 (a) presents the message delivery's reliability performance. For all the tested background traffic bandwidths, our scheme can correctly detect at least 99% of them. Figure 22 (b) shows the accuracy performance. Without using the self-correcting encoding/decoding algorithm, the accuracy ratio decreased as the background traffic bandwidth increased. For the saturated background traffic bandwidths, the accuracy percentage was 92%. Similar to the uncontrolled experiment, we analyzed all the wrongly decoded messages and marked those that were correctable. After applying the self-correcting algorithm, the accuracy percentages for all the background traffic bandwidths increased to at least 98%.

Message delivery overheads. To evaluate the message delivery overheads imposed on network throughput, we tested the iperf UDP bandwidth between two *directly* connected WiFi nodes while a HoWiES AP was sending WiFi-ZigBee messages with different frequencies in vicinity. We have tested the message sending frequencies (Hz) of 0.5, 1, 2, 5, 10, 20, $\cdots$, 60, 80 and 100. Figure 23 (a) shows the experiment result. With the message sending frequencies (Hz) of 0.5, 1, and 2, there were only a negligible amount of throughput degradation on network throughput. With the sending frequencies of 5 and 10, the tested iperf

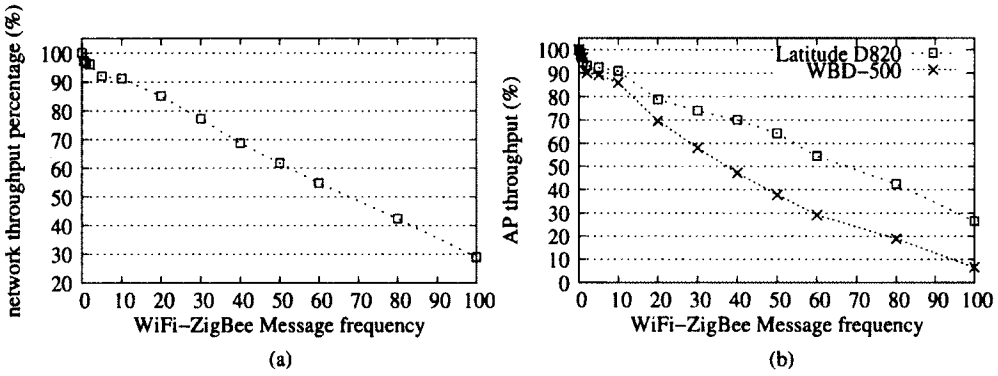


Figure 23: HoWiES WiFi-ZigBee message delivery overheads. (a) shows the message delivery overheads on network throughput. (b) shows the message delivery overheads on performances of two different AP platforms: Latitude D820 laptop and WBD-500 standalone AP.

connection still had 90% of its bandwidth. Then the network bandwidth decreased approximately in linear as the message sending frequency increased.

To evaluate the overheads imposed on AP performances, we established an iperf UDP connection between two WiFi node *via* a HoWiES AP. Then we tested the bandwidth between the two WiFi nodes while the HoWiES AP varied the WiFi-ZigBee sending frequencies in the same way as in the previous network overhead experiment. We tested our implementation on two different AP platforms: the Dell Latitude D820 laptop and the Wiligear WBD-500 standalone AP. Figure 23 (b) shows the experiment result. Similar to the network overhead experiment, both AP platforms has a small amount of throughput degradation when the message sending frequency is smaller than 10. When the sending frequency is higher than 10, the throughputs on both platforms decreased linearly as the message sending frequency increased. The WBD-500 standalone AP had a faster performance drop than the Dell laptop. This is because the standalone AP has more constrained computational resources.

From the two experiments we learn that our message delivery scheme has a negligible amount of overheads on both network throughput and AP performance when the message sending frequency is less than 10 (i.e., 10 messages per sec-

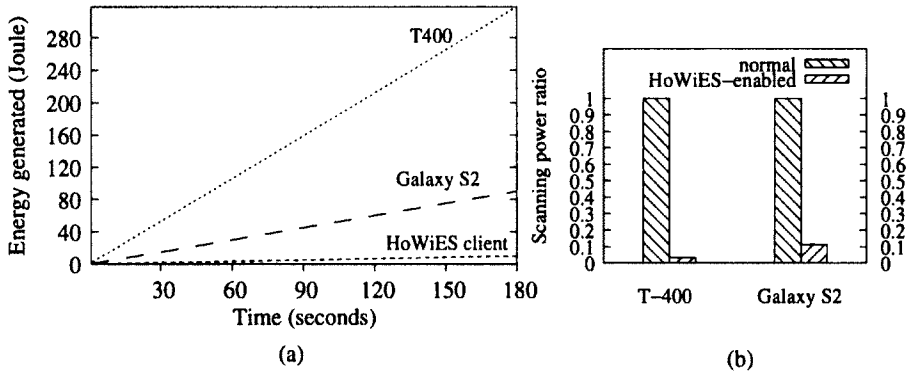


Figure 24: Energy gain on the WiFi scanning state. (a) is the energy generated by the WiFi scanning operation as the time elapses. (b) is the scanning power consumption comparisons between the two types of HoWiES clients and corresponding normal devices.

ond). When the sending frequency is higher than 10, the overheads increase linearly as the frequency increases.

3.4.2 Energy gain achieved by the energy saving protocols

Power measurement setup and methodology. To measure the power consumption in the T400 laptop, we use the smart battery interface come with the operating system. According to [28], the smart battery interface is highly accurate when the battery interface reading rate is low. Since we are only interested in long term energy consumptions, the smart battery interface satisfies our requirements. To measure the power consumption in the smartphone, we use the Monsoon power monitor [73], which provides accurate power readings for handheld mobile devices. When we measure the power of a device, we turn off all the unnecessary applications and services, and shut down the power-hungry LED screen. To get the power consumption value for a WiFi operation (e.g., scanning or standby) in a device, we first measure the baseline system power consumption (i.e., system power consumption without running any WiFi operations). Then we measure the system power when the device is continuously performing the targeted WiFi operation. Finally, the difference between the two values is the power consumption for the WiFi operation.

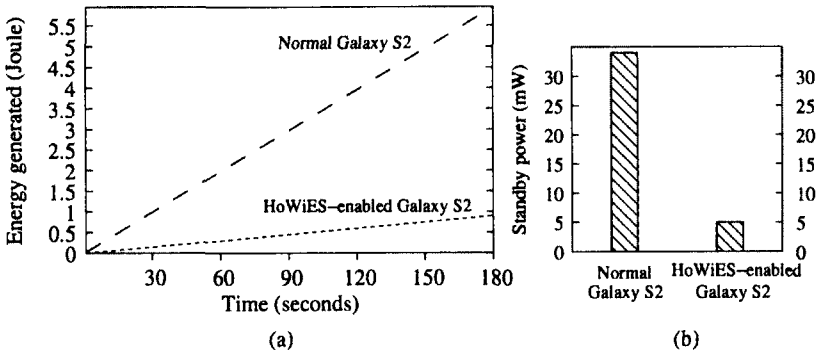


Figure 25: Energy gain on the WiFi standby state. (a) is the energy generated during standby as the time elapses. (b) is the power consumption comparison between the HoWiES-enabled Galaxy S2 smartphone and the original device.

Energy gain in WiFi scanning. We measured the WiFi scanning power consumptions of three devices: a normal T400 laptop, a normal Galaxy S2 smartphone and a HoWiES client. Our measurement shows that the T400 laptop, the Galaxy S2 smartphone and the HoWiES client spend 1740 mW, 501 mW and 61 mW for WiFi scanning respectively. Figure 24 (a) shows the energy generated by the WiFi scanning operation as the time elapses in a 3 minutes duration. Figure 24 (b) shows the percentages of WiFi scanning power reduction of the HoWiES client when compared to the normal mobile devices. From the result we can conclude that our scheme can effectively reduce power consumptions for the WiFi operation in mobile devices.

Energy gain in WiFi standby. To evaluate the power savings achieved in the WiFi standby state, we compared a Galaxy S2 smartphone and its HoWiES-enabled version. Our measurement shows that the normal Galaxy S2 and the HoWiES-enabled Galaxy S2 consumes 33 mW and 5 mW in the standby state respectively. Figure 25 (a) shows the energy generated during standby as the time elapses in a 3 minutes duration. Figure 25 (b) compares the standby power consumption between the two subjects. Although the absolute value of power consumption gain is small at the first glance, it is still quite meaningful considering that users usually leave the WiFi radios in their mobile devices idle most of the

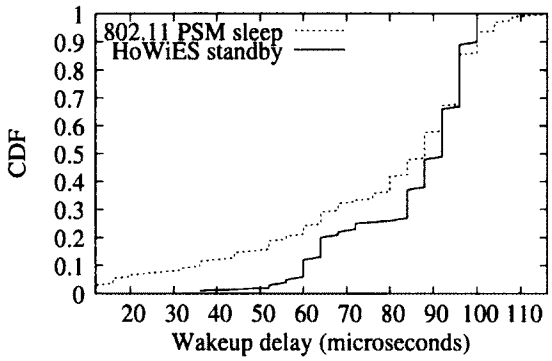


Figure 26: Empirical HoWiES wakeup delay CDFs of a normal Galaxy S2 and a HoWiES-enabled Galaxy S2.

time.

3.4.3 HoWiES wakeup delay

We evaluate the delay performance of our implemented system in terms of waking up a standby client. To do the evaluation, we instrumented the WiFi device driver in AP to record the times that a 802.11 PSM standby client and a HoWiES-standby client needs to wake up and get their buffered packets: when the first incoming packet of a standby client is enqueued, the AP records the packet enqueue time T_s and wakes up the standby client to get its packets (through either standard 802.11 PSM wakeup operation or HoWiES wakeup operation). The AP records the time T_e when the client notifies the AP that it is ready to receive the buffered packets. The time that the client used to perform the wakeup operation is calculated as $T_e - T_s$. On the clients side, the wakeup interval of the normal Galaxy S2 is set to a beacon period, which is the default setting used by the WiFi driver. For the HoWiES-enabled Galaxy S2, it goes to sleeping state once it enters HoWiES standby, and keeps sleeping until it is waken up by a WiFi-ZigBee message. Figure 26 shows the empirical CDF of time that a normal Galaxy S2 and a HoWiES-enabled Galaxy S2 needs to wake up. Through the figure we can see that the wakeup delay of our implemented system is already comparable to that of a normal 802.11 PSM client. Actually there is still room to improve the

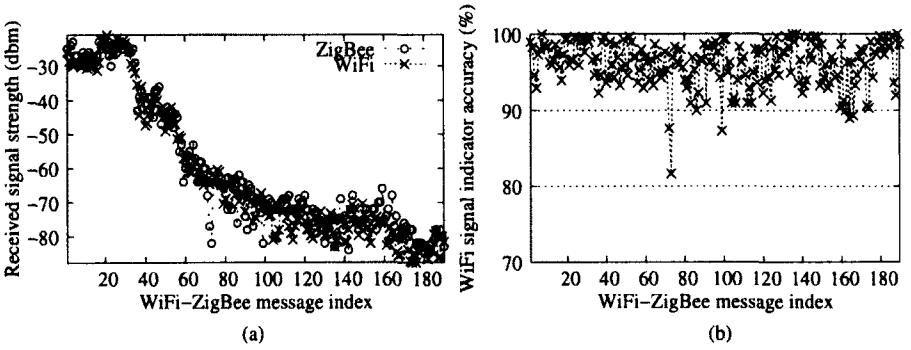


Figure 27: Accuracy of ZigBee based WiFi signal strength indicator.

wakeup latency in our implementation. For example, currently an AP is using a user space program to transmit message packets. This will incur some extra time in the kernel-user space communication. Moreover, the user space program cannot set its packets to have higher transmission priority than other packets, which may cause more extra time between two message packets.

3.4.4 WiFi signal strength indicator by using ZigBee

When a HoWiES client in scanning state detects a WiFi network advertisement WiFi-ZigBee message, the HoWiES manager uses the signal strength indicator (SSI) generated by the ZigBee radio to determine the signal quality of the WiFi network. In this experiment, we evaluate how accurate the WiFi SSIs generated by ZigBee radios are when compared to SSIs that are generated by a WiFi radio. In the experiment, we let a HoWiES AP transmit WiFi-ZigBee messages continuously with an interval of 1 second. Then we walked further away from the HoWiES AP carrying a HoWiES client, which ran a program that captured all the WiFi-ZigBee message packets using a WiFi sniffer while the ZigBee radio was continuously detecting, decoding and recording messages. To process the data, we first correlated the 3 WiFi packets (recall that each WiFi-ZigBee messages is encoded by 3 WiFi packets) with the corresponding WiFi-ZigBee message. The SSI of each WiFi-ZigBee message generated by the ZigBee radio is calculated as

the average of all the positive energy samples of the message, and the SSI generated by the WiFi radio is the average of the 3 corresponding WiFi packet's SSI. We plot the SSI values generated for each message by both radios in Figure 27 (a), and plot the accuracy percentage of each SSI generated by the ZigBee radio compared to the corresponding WiFi generated SSI. From the figure we can see that ZigBee generated SSI can accurately reflect the actual WiFi signal strength.

3.5 Conclusion

HoWiES is a Wifi energy saving system that achieves WiFi energy savings in three different aspects: scanning energy saving, standby energy saving and standby wakeup contention reduction. The foundation of the HoWiES system is a novel WiFi-ZigBee message delivery scheme that enables WiFi radios to deliver different information to ZigBee radios. Our extensive evaluations show that our implementation of the WiFi-ZigBee message delivery scheme works accurately and reliably with reasonable overheads, and that the whole system can effectively save energy for WiFi devices.

4 CacheKeeper: A System-wide Web Caching Service for Smartphones

Smartphone applications that need connectivity usually rely on certain application layer data transmission protocols to exchange data. HTTP is one of such protocols that are being used by many smartphone applications. HTTP traffic is now the dominant type of Internet traffic [16]. With the popularity of smartphones and tablets, an increasing amount of HTTP traffic originates from mobile devices. The mobile HTTP traffic has grown 35% in under a year [17], and now accounts for 20% of the U.S. HTTP traffic [18]. However, we found that many smartphone applications incur unnecessary energy consumption by issuing redundant web transmissions. In this project, we analyzed the reason for the unnecessary energy consumption, designed and implemented CacheKeeper, a system-wide web caching service to solve the problem.

4.1 Background and related Work

4.1.1 Background

Unlike conventional PCs, where web browser is the main source of web traffic, smartphones have another significant source of web traffic: dedicated mobile apps. The popularity of the ubiquitous smartphone is partly driven these useful and entertaining mobile apps, all available for little or no cost. Since most mobile

apps utilize some form of network connectivity, the network behavior of mobile apps is an important area of research.

An appropriate web caching implementation in mobile apps will benefit both users and network operators. With such an implementation, users can (a) experience a higher quality of service, since the data can be accessed faster locally, (b) lower costs, since users may have to pay a higher fee for downloading more data, and (c) conserve energy by reducing unnecessary data transmissions. Network operators also benefit when mobile apps implement web caching correctly since this reduces the congestion on the network, especially the last mile radio connections.

Despite the importance of web caching, large numbers of mobile apps have *imperfect web caching*, meaning that web caching is either implemented for only certain HTTP resources the apps request, or is not implemented at all. The reason is twofold: lack of library support and negligence from developers. For example, the Android platform provides two official HTTP client classes: `HttpURLConnection` and `Apache HTTP Client` [74]. Before Android 3.2 (API level 13), the `HttpURLConnection` class only provided an interface for caching implementation. Developers have to implement their own client-side caching mechanisms. Heavy programming burden will hold developers from doing so. Later, Android added an official implementation of client-side caching (i.e., the `HttpResponseCache` class) for `HttpURLConnection`. However, it still requires developers to call the library to add caching capability. Since apps without caching or with poor caching will still have the "look-and-feel", some developers will spend less time implementing and testing the caching behavior of their apps.

4.1.2 Related work

Measurements of Web Usage in Smartphones. The popularity of smartphones and tablets has driven a growing number of works on studying web usage in smart-

phones. Based on a dataset containing one-year-long web accessing log from 24 iPhone users, recent work [75] studies users' Internet accessing behaviors on smartphones. The study results show that dedicated mobile apps are used by users to visit the web much more frequently than browsers. This demonstrates the needs to ensure properly working web functions, including web caching, for mobile apps. Work [76] specifically investigates smartphone web traffic related to advertisements based on a large dataset collected in a major European mobile network. The results suggest that ad traffic is a major component of overall mobile web traffic. Work [77] compares smartphone web traffic and laptop web traffic based on a 3-week-long wireless communication trace collected in an enterprise environment. As one of the findings, the authors suggest that web caching in smartphones is not as effective as that in laptops. Similar to [77], Qian *et al.* [78] conduct a comprehensive measurement study on web caching in smartphones. By examining a one-day smartphone web traffic dataset collected from a cellular carrier and a five-month web access trace collected from a small user base, the study reveals that about 20% of the total web traffic examined is redundant because of poor web caching. In this work, we investigate the effectiveness of web caching in smartphones from a different perspective. Instead of analyzing mobile web traffic collected from service provider, we inspect web caching function of 1300 top ranked apps downloaded from the Google Play. This way, we can explicitly get, rather than inferring, information about how different types of mobile apps perform in web caching, which we believe will be helpful for future mobile apps and mobile platforms design.

Reducing Web Accessing Latency in Smartphones. A considerable amount of efforts have been invested in reducing web accessing latency in smartphones. To increase the operation speed of web browsers, work [79] proposes improved web caching on style/layout data. Work by Wang *et al.* [80] also studies the causes of slow web mobile browsers. The authors suggest the root cause is slow con-

tent loading. They then propose a method of speculative loading [81] to reduce web accessing latency when using smartphone browsers. PocketSearch [82] proposes to put results of certain cloud service like web search in smartphones' local storage to expedite service speed. Similarly, PocketWeb [83] proposes, using machine learning on a per users basis, to prefetch web pages into smartphone's local storage to reduce web accessing latency. In this work, we take a different approach to reduce web accessing latency for smartphones. We propose to run web caching as a system service, so that we can compensate for the flaw of imperfect web caching in many mobile apps, which causes unnecessary transfers, increases web accessing latency and reduces battery life.

4.2 Motivation

Our approach is to reduce the burden of mobile app developers by providing a caching-as-a-service layer. The web caching service will provide the correct web caching implementation with no effort on the part of mobile app developers. Developers do not need to install any additional libraries or incorporate any additional API calls to take advantage of CacheKeeper. There are two major observations that led us to believe that it is desirable to provide web caching as a system-wide service for smartphones: *web caching imperfection in mobile apps* and *cross-app caching opportunities*.

4.2.1 Web caching imperfection in mobile apps

We have conducted an extensive measurement study of top-ranked Android apps in Google Play to study the web caching behaviors of individual Android apps.

Measurement setup

Apps selection. The Google Play organizes apps into 24 categories (shown in first column of Table 7). We downloaded the top 50 ranked free apps from each of category, except the "News and Magazines" category. In this category, we selected the top 150 ranked free apps. We paid more attention to the news apps because they all access web contents. In total we selected 1300 top ranked apps. We also inspected the selected apps to ensure no app appears in two different categories.

Web traffic generation. We installed and used each app on a smartphone running Android 4.0 to see if the app generates web traffic. To achieve automated testing, we developed a tool (using the `ADB getevent/sendevent` utility) that can record and replay user inputs on the touch screen. Prior to running the automated measurement experiment, we first recorded the user inputs when we used an app. To ensure comprehensive app usage, we clicked all the representative buttons/tabs/links when recording the user inputs. During the measurement experiment, we replayed the recorded user inputs to test all the 1300 apps. The experiment has been run twice with a one-week interval between the two executions.

Web traffic recording. During the measurement experiment, we configured the smartphone to access the Internet via an HTTP debugging proxy [84], through which we could capture all the HTTP traffic the smartphone generated. The captured HTTP traffic was saved into trace files for later processing. Among the 1300 apps, there are 863 apps generating HTTP traffic. Table 7 column `I.2` shows the number of apps with HTTP traffic for each category. To quantify how much HTTP traffic an app generates, we computed the per-click HTTP traffic volume for each app, which is the ratio of an app's total HTTP traffic volume over the app's total number of clicks. Table 7 column `I.3` shows the average per-click HTTP traffic

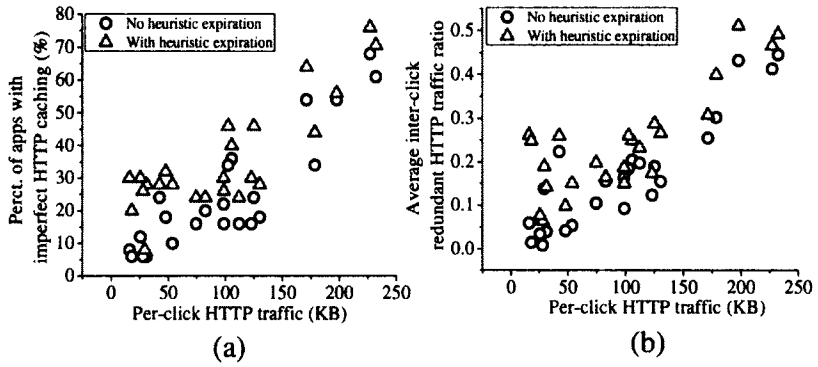


Figure 28: Correlation between per-click HTTP traffic and (a) the number of apps with imperfect web caching, and (b) the average inter-click redundant HTTP traffic ratio.

volume for each category.

Web caching imperfection identification. When testing an app, we executed the app twice by replaying the user inputs twice with an short interval, and collected traces for the two executions. We chose a short execution interval because we wanted to ensure that the cacheable HTTP objects (defined in RFC 2616 [85]) obtained in the first execution are still fresh when the second execution happens. If the second trace contained the same cacheable HTTP objects as in the first one, and the cacheable objects in the first one were still fresh when the second execution occurred, then the app would be identified to have imperfect web caching, and the corresponding HTTP transaction (i.e., the HTTP request/response pair) in the second trace would be labeled as redundant. For an HTTP response that does not contain expiration time or validators (e.g., ETag, Last-Modified time), if it neither contains the `Cache-Control:no-store` directive, we treat it as *heuristic cacheable* (because in this case, according to RFC2616, HTTP caches can assign a heuristic expiration time to the response).

Measurement findings

App HTTP traffic and web caching imperfection. Figure 28 (a) plots, for the 24 categories of apps, the relationship between each category's per-click HTTP traffic and the category's percentage of apps with imperfect web caching. We

Table 7: Summary of the app measurement study.

Categories on Google Play	I: Setup			II: Inter-click. redundancy	
	1	2	3	1	2
	Apps tested	Has HTTP traffic	HTTP traf. per-click	Apps cnt.†	Traf. ratio†
books & refs	50	30	42.3 KB	12|14	0.23|0.26
business	50	23	16.0 KB	4|15	0.06|0.26
comics	50	39	125.1 KB	12|23	0.19|0.29
communication	50	17	18.0 KB	3|10	0.02|0.25
education	50	31	130.3 KB	9|14	0.16|0.27
entertainment	50	37	105.7 KB	18|20	0.21|0.25
finance	50	16	29.4 KB	3|4	0.14|0.19
health & fitness	50	35	74.4 KB	8|12	0.11|0.20
libs & demos	50	29	82.7 KB	10|12	0.16|0.17
lifestyle	50	30	98.8 KB	8|13	0.10|0.15
media & video	50	37	122.9 KB	8|15	0.13|0.18
medical	50	33	31.1 KB	3|14	0.04|0.15
music & audio	50	28	98.6 KB	11|15	0.17|0.19
news & mgzns	150	129	232.2 KB	92|106	0.45|0.50
personalization	50	34	53.5 KB	5|14	0.06|0.16
photography	50	40	47.6 KB	9|16	0.05|0.10
productivity	50	26	27.7 KB	3|13	0.01|0.07
shopping	50	34	197.7 KB	27|28	0.44|0.52
social	50	17	112.1 KB	8|12	0.20|0.24
sports	50	44	227.2 KB	34|38	0.42|0.47
tools	50	40	25.6 KB	6|15	0.04|0.08
transportation	50	40	102.6 KB	17|23	0.19|0.26
travel & local	50	29	178.5 KB	17|22	0.31|0.40
weather	50	45	171.2 KB	27|32	0.26|0.31
Overall	1300	863	121.2 KB	354|500	0.19|0.26

†: Presented in the format of $a|b$, where a and b are the values without and with heuristically redundant traffic counted respectively.

can see that the ratio of apps with imperfect web caching in a category is roughly proportional to the category's average per-click HTTP traffic. We can also learn that almost all the (four out of five) categories whose per-click HTTP traffic is greater than 150 KB have more than half apps with imperfect web caching. This suggests that *imperfect web caching is a common among apps with high HTTP traffic volumes.*

Inter-click HTTP traffic redundancy. We label a redundant HTTP transac-

Table 8: Summary of the app measurement study (continued).

Categories on Google Play	III: Same-click redundancy		IV: Advertisement			
	1	2	1	2	3	4
	Apps cnt.†	Traf. ratio†	Apps cnt.	Ad only	Ad traf. per-click	Cacheable traf. ratio†
books & refs	3|3	0.04|0.04	19	10	10.3 KB	0.77|0.87
business	1|1	0.01|0.01	14	6	5.1 KB	0.92|0.93
comics	3|6	0.03|0.04	31	18	13.0 KB	0.68|0.89
communication	2|2	0.01|0.02	9	4	11.4 KB	0.61|0.94
education	2|2	0.01|0.01	24	15	13.7 KB	0.88|0.89
entertainment	3|4	0.04|0.05	29	4	32.0 KB	0.89|0.92
finance	0|0	0|0	14	7	15.7 KB	0.87|0.88
health & fitness	2|3	0.04|0.06	33	13	37.5 KB	0.86|0.94
libs & demos	4|4	0.04|0.04	24	16	19.9 KB	0.81|0.95
lifestyle	1|1	0.01|0.01	24	5	12.5 KB	0.70|0.78
media & video	6|8	0.03|0.04	32	16	38.0 KB	0.94|0.97
medical	0|0	0|0	30	20	16.6 KB	0.85|0.91
music & audio	4|5	0.04|0.04	25	6	39.0 KB	0.87|0.93
news & mgzns	41|45	0.12|0.12	94	7	39.4 KB	0.85|0.89
personalization	1|1	0.01|0.01	33	16	27.9 KB	0.81|0.88
photography	1|2	0.01|0.01	37	12	23.6 KB	0.91|0.93
productivity	1|2	0.01|0.01	22	10	12.5 KB	0.80|0.82
shopping	11|13	0.11|0.15	20	1	17.4 KB	0.94|0.97
social	1|1	0.01|0.01	16	3	30.7 KB	0.87|0.95
sports	17|18	0.09|0.09	31	3	18.1 KB	0.82|0.86
tools	0|3	0|0.01	36	21	19.4 KB	0.78|0.84
transportation	2|2	0.03|0.03	32	7	37.0 KB	0.88|0.92
travel & local	4|5	0.02|0.02	23	4	7.9 KB	0.61|0.89
weather	11|12	0.03|0.03	43	6	30.7 KB	0.76|0.83
Overall	121|143	0.03|0.04	695	230	24.2 KB	0.84|0.90

†: Presented in the format of $a|b$, where a and b are the values without and with heuristically redundant traffic counted respectively.

tion as *inter-click redundant* if the original transaction and the redundant transaction occur as results of two different clicks on the same app. Table 7 column `II.1` shows the number of apps with inter-click redundant HTTP traffic for each category. We calculate the inter-click redundant traffic ratio of a category as the ratio of the category's total inter-click redundant traffic over its total HTTP traffic. Table 7 column `II.2` shows this value of each category. The inter-click redundant traffic ratio is 0.19 for all the apps tested. This number increases to 0.24 when

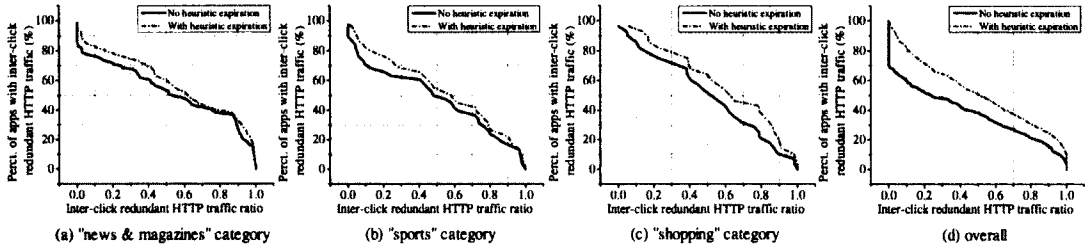


Figure 29: Distribution of inter-click redundant traffic ratio: (a)-(c) show the CCDF of the redundant ratio for the apps with inter-click redundant traffic in the top 3 categories with the most per-click HTTP traffic; (d) shows the same statistics for all the imperfect apps.

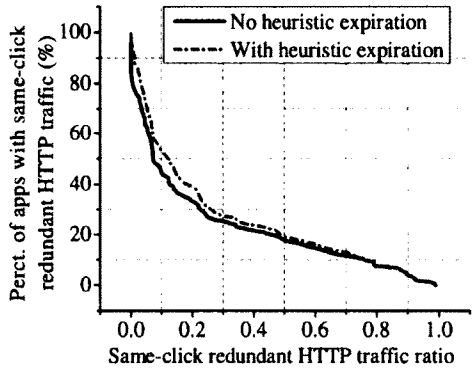


Figure 30: CCDF of the same-click HTTP redundant traffic ratio.

counting heuristically redundant traffic. Figure 28 (b) plots, for the 24 categories, the relationship between each category's per-click HTTP traffic and its inter-click redundant traffic ratio. We can observe that those categories with high per-click HTTP traffic have much higher inter-click redundant traffic ratios. For example, the inter-click redundant traffic ratios for the top 3 categories with the most HTTP traffic are 0.45 (News & Magazines), 0.42 (Sports) and 0.44 (Shopping). To further study the distribution of the inter-click redundant traffic ratio among apps, we plot in Figure 29 the CCDFs of the inter-click redundant traffic ratio for the previous three categories and for all the apps tested. From the figure we can learn that for the top three categories with the most per-click HTTP traffic, half of the apps with inter-redundant traffic have a redundant ratio greater than 0.5, which suggests *imperfect web caching is not only a common, but also a serious flaw for apps with high HTTP traffic volumes.*

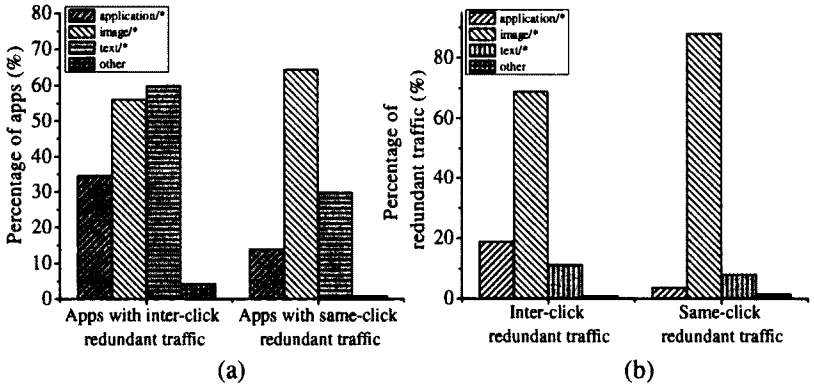


Figure 31: Content type breakdowns for (a) number of apps and (b) redundant HTTP traffic.

Same-click HTTP traffic redundancy. We found that a notable amount of apps we tested downloaded the same resource multiple times for the same user click. We call those redundant HTTP transactions occur for a single click on the app as *same-click redundant* HTTP transactions. Table 8 column III.1 and column III.2 list, for each category, the number of apps with same-click redundant HTTP traffic and the same-click redundant HTTP traffic ratio. Overall, about 10% of the apps have same-click redundant HTTP traffic, and the average same-click redundant traffic ratio is 0.03. However, similar to the case of inter-click HTTP traffic redundancy, these two figures are much higher for those categories with high HTTP traffic volumes. For example, for the top three categories with the most HTTP traffic, more than 20% of the apps have same-click redundant HTTP traffic, and the traffic ratio is around 10%. We plot the CCDF of the same-click redundant ratio in Figure 30, which shows that about 40% of all the apps with same-click redundant HTTP traffic have a redundant ratio greater than 10%.

By carefully examining the web contents that involved same-click redundant HTTP transactions, we confirmed that those redundant downloads for the same click were not because the same resources needed to be displayed at several places on the same web page. We believe the main cause for same-click redundant HTTP transactions is developer error. As an evidence, a well-known online

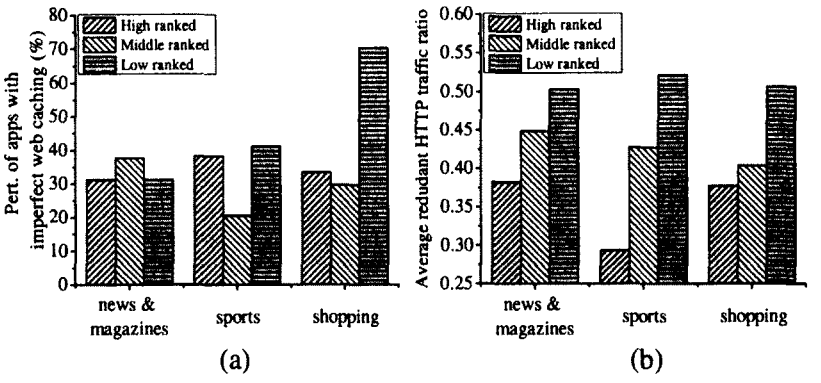


Figure 32: Web caching imperfection and app rankings.

shopping and auction app had a self-redundant traffic ratio of 0.64 for the version we tested, and the problem was fixed in a new version when we retested the app several months later.

Content types of redundant HTTP traffic. By extracting the `Content-Type` field from the HTTP response headers, we identified three major types of HTTP resources appeared in the measurement experiment: `application/*`, `image/*` and `text/*`. Figure 31 (a) shows for all the apps with redundant HTTP traffic, the percentage of apps neglecting to cache each type of HTTP resources. In the figure, all the types other than the three major types are labeled as `other`. According to our experience, many of the apps with redundant traffic on image resources only cache large images, but fail on caching small images like thumbnail images for news lists. Meanwhile, almost all the apps with redundant traffic on text resources fail to cache all kinds of text objects such as configuration files and data files. Figure 31 (b) shows the content type breakdown for the redundant HTTP traffic. We can learn that image resources took the most redundant traffic. In the meantime, text resources also account for about 10% of all the redundant HTTP traffic.

App ranking and web caching imperfection. We have investigated whether app rankings have relationship with imperfect web caching. For the top 3 categories with the most per-click HTTP traffic we tested, we divide their apps into

three groups (i.e., high, middle and low ranked) according to the app rankings by on Google Play. We plot the percentage of apps with imperfect web caching and the average redundant HTTP traffic ratio of each group in Figure 32 (a) and (b) respectively. The shopping category has a much higher percentage of imperfect apps in the low ranked group. Meanwhile, for all the three categories, there is a clear increasing trend for redundant traffic ratio from the high ranked group to the low ranked group. Thus, we can cautiously make an conclusion that apps with lower ranking are more likely to have poor web caching implementation. This is reasonable because high ranked apps are usually developed by experienced and well-known developers, who are more likely to pay attention to details like web caching for their apps.

4.2.2 Cross-app caching opportunities

Same-app web caching reduces web accessing latency and saves bandwidth for an app when it access the same cacheable content more than once. Meanwhile, cross-app web caching can also achieve the same benefit for different apps accessing the same web content. We have identified two types of cross-app caching opportunities specially for mobile apps.

Opportunities by user behaviors. The first type of opportunities comes when a user uses different apps to access the same web content. For example, many top-ranking news reader apps on Google Play (such as Flipboard [86], Pulse [87] and Yahoo! [88]) provide a function to let users view the news they are browsing on phone's web browser. This is a useful feature because usually a web browser provides more full-fledged web content rendering support. With this feature, users may access the same piece of news several times with both the news reader app and a web browser. Another example is that when a user wants to do online shopping with his smartphone, he may first uses a web browser to search for the product and compares prices and reviews. After seeing that an online

retailer, Amazon.com for example, provides the product for the lowest price, the user opens Amazon's dedicated shopping app to complete the transaction.

Opportunities by shared libraries. The second type of opportunities comes when two different apps use the same shared library that regularly accesses web contents. Mobile advertising network SDKs are the most notable ones of such kind of shared library. The way that a developer puts ads in his app is to call functions from an ad library provided by the mobile ad network. The app will download (or the ad network will push) advertisements to the smartphone running the app dynamically when the app is being used. Mobile ads are common in free mobile apps. For example, among the 1300 apps in our measurement experiment, 695 apps generate ad HTTP traffic (Table 8 column `IV.1`); HTTP traffic of 230 apps are all ad traffic (Table 8 column `IV.2`); and the per-click ad traffic is 24.2 KB (Table 8 column `IV.3`), which accounts for 20% of the per-click HTTP traffic. In the mean time, most of the ad traffic is cacheable: as shown in Table 8 column `IV.4`, the overall cacheable ad traffic ratio is 0.84 (or 0.9 if considering heuristic expiration). Considering that the mobile ads market is dominated by just a few ad networks [76] and that the ads to be shown are usually determined based on the user information such as user's location [89], it's likely that different apps running on the same phone will display the same set of ads over time. According to our experience, even two different ads from the same ad network usually share common cacheable objects like configuration scripts and data files.

4.3 System design

4.3.1 Design goals and challenges

Design goals. We design CacheKeeper (CK for short) with the following goals in mind.

1. CK should be able to perform standard-compliant (RFC 2616 [85]) web caching for all the entities (e.g., apps) making HTTP requests in the device. This is the fundamental goal of designing CK.

2. CK should be transparent to all the entities that it serves. In other words, entities making HTTP requests should be able to perform normally without any modifications. This is to ensure backward-compatibility for exiting apps.

3. Since CK is essentially a shared client-side cache, the design of CK should provide means to protect apps' cache privacy.

4. While cache hits will bring benefits, CK should also incur low overhead on cache misses to ensure good usability.

5. CK should provide interfaces allowing users to configure the web caching services (e.g., cache size, cache location and heuristic expiration time) and to obtain service status.

Challenges. The design of a client-side system-wide caching service such as CK is different from implementing a cache in an individual app and implementing a proxy cache.

In particular, when compared with app-based client cache, there are two challenges: The *first* challenge is the ability to handle a large volume of concurrent HTTP transactions while incurring low overhead. This is different from caching in individual apps where HTTP requests are issued less frequently and usually in a sequential manner. The *second* challenge is that, unlike individual apps where web caching is part of the operations handled by HTTP libraries, CK is not in the network operations flow of the apps it serves. Thus, it is challenging to design and implement CK without making any modifications to the apps. For example, since fetching content from web cache is fast, it is designed as a synchronous operation in individual apps (i.e., the program execution blocks until the fetching operation

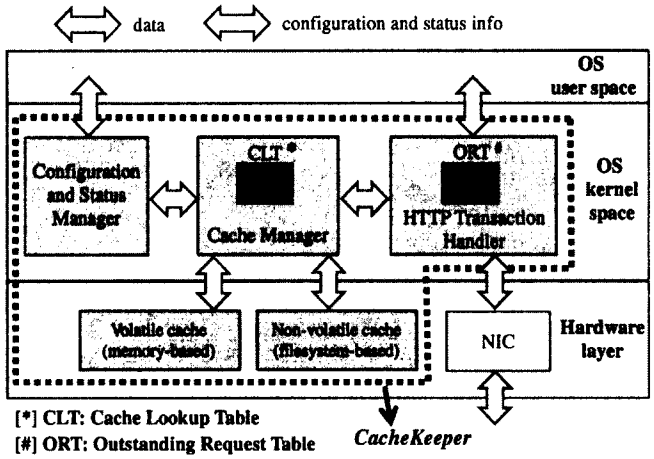


Figure 33: CacheKeeper architecture.

finishes). However, fetching content from web cache cannot be designed as a simple synchronous operation in CK. This is because apps use asynchronous requests (i.e., request-then-poll) to retrieve content from web servers. Acting as a transparent middle layer between apps and web servers, CK cannot serve asynchronous requests from apps by using simple synchronous web cache retrieving. Otherwise it will be extremely inefficient and unscalable.

The differences between designs of CK and proxy cache originate from their operation contexts. Since CK is a system service serving apps running in the user space, its design focus is twofold: maintaining the transparency to the apps while not sacrificing caching performances, and protecting apps' cache privacy. Unlike CK, a proxy cache does not need to consider apps' contexts and privacy. Since a proxy cache needs to serve thousands of computers on the network, the design focus of it is cache replacement algorithms [90], which plays less important role in CK's design.

4.3.2 CacheKeeper architecture

The architecture of CK is shown in Figure 33. Cach-eKeeper is designed as an OS kernel space component providing web caching service to apps running in the user space. We choose to place CK in OS kernel space for three reasons. *First*, this

SK: socket of the HTTP transaction **RU**: requested URL
ICH: is cache-hit (False by default) **IP**: is private (False by default)
CRA: cached response address

SK	RU	ICH	IP	CRA

(a) Entry of the outstanding requests table (ORT)

RU: requested URL **VCE**: volatile-cache entry
NVCE: non-volatile-cache entry

RU	VCE	NVCE

(b) Entry of the cache lookup table (CLT)

Figure 34: Entry structures of ORT and CLT.

approach has clear performance advantage over user space based approaches (e.g., user-level HTTP proxy). *Second*, this allows the web caching service to be *portable* across different devices running the same type of OS kernel. *Third*, by placing it in kernel space and not changing the interfaces connecting user and kernel spaces, we could easily achieve *backward-compatibility*.

CK contains the following components: the HTTP transaction handler, the cache manager, the configuration and status manger and the physical caches. Next, we give a description of each component, followed by the description of how the components cooperate in CK operations.

HTTP transaction handler. The HTTP transaction handler handles HTTP requests issued from apps and HTTP responses retrieved from network connections. The transaction handler consults the cache manager for cached responses, and passes incoming responses to the cache manager for caching processing.

The transaction handler uses a key data structure, *outstanding requests table* (ORT), to handle the asynchronous web content requests from apps mentioned previously. Each ORT entry corresponds to an HTTP request waiting to be served. Figure 34 (a) shows the ORT entry structure. Since CacheKe-eper needs to process a large amount of concurrent HTTP transactions, we use socket address plus the requested URL to identify individual HTTP transactions. The `SK` field and the `RU` field record the socket address and the requested URL of the corresponding HTTP transaction respectively. The `ICH` field records if CK has a fresh cached

response for the HTTP request. The IP field records whether the app issuing the request has declared that the HTTP transaction is private and thus should not be cached by CK. The default values of both ICH and IP are False. If the request can be served by CK, the field CRA holds the address of the buffer that is prepared by the cache manager and stores the cached response.

Cache manager. The cache manager performs the following tasks. It accepts and processes queries for cached response from the transaction handler. It accepts newly coming responses from the transaction handler, caches them in a proper physical cache, and performs cache replacement if necessary. It accepts and processes configuration or status query requests from the configuration and status manager.

To help manage the cache entries, the cache manager maintains a key data structured named *cache lookup table* (CLT). Each CLT entry, with the entry structure shown in Figure 34 (b), corresponds to the cached HTTP transaction (i.e., a cached HTTP request/response pair) of a certain URL. The field RU records the URL of the cached transaction. The fields VCE and NVCE store the addresses of the volatile cache entry (i.e., memory-based) and the non-volatile cache entry (i.e., filesystem based) of the HTTP transaction respectively.

Configuration and status manager. The configuration and status manager provides interfaces to user space programs to configure CK and to query the running status of CK for debugging purposes.

Physical caches. CK supports two types of caching media: *volatile cache* residing in device's memory and *non-volatile cache* residing in device's filesystem. The volatile cache is for efficient cache lookup, and the non-volatile cache is to ensure persistent cache content after reboots.

4.3.3 CacheKeeper in operation

On cache hits/misses/validations. The transaction handler handles every HTTP request from apps and HTTP response from the network. Upon receiving an HTTP request, the transaction handler creates a new ORT entry, and consults the cache manager to see if CK has a freshed cached response for the request. The cache manager looks up the CLT by comparing the URL provided by the transaction handler and the RU field in the CLT entries, and sends the result back to the transaction handler, which in turn updates the ICH field in the ORT entry based on the result. *If there is a cache hit*, the cache manager retrieves the cached response from either the memory based cache or the filesystem based cache based on the VCE and the NVCE fields in the CLT entry, and notifies the transaction handler about the address of the buffer storing the cached response. The transaction handler then records this address in the CRA field of the ORT entry. Till now, the transaction handler has the complete ORT entry, through which the handler knows how to serve the later polls from the app (recall that apps use request-then-poll to retrieve contents from web servers). *If there is a cache miss or the cache response is expired*, the HTTP request is sent out as normal. The transaction handler passes the HTTP response to the cache manager for storing if the response is cacheable. *If the cached response needs to be validated* before it can be served to the apps, the transaction handler uses the validator (e.g., ETag, Last-Modified time) provided in the cached response to issue a conditional request to the web server. Based on the result of the conditional request, the following operations are similar to the cache hit or miss situation described previously .

Dealing with same-click redundant requests. During the measurement experiment, we observed that a notable amount of apps generated same-click redundant HTTP traffic. CK will naturally eliminate same-click redundant traffic if

it has cached the previous response for a redundant HTTP request. However, a deeper investigation into the same-click redundant HTTP transactions we obtained shows that HTTP requests of about 20% of the same-click redundant transactions were issued before the full responses of the first HTTP transactions were received, in which case CK would send out those redundant requests. To solve this problem, the transaction handler postpones sending out an HTTP request for a short period of time if the requested URL is found in an ORT entry. Based on our experience, we set the length of this period to 200 ms in our prototype implementation.

Declaring private HTTP transactions. As one of the design goals, CK should provide means to protect apps' cache privacy. In our design, we allow apps to decide if they want their HTTP traffic to be cached by CK. Specifically, we provide an interface for apps to declare privacy for each HTTP transaction they generate. If an HTTP transaction is declared as private, it will not be cached by CK. We will present the implementation of the interface later.

4.4 System implementation

We have implemented a prototype of CK as a loadable Linux kernel module (kernel version: 3.0.15).

Location in Linux kernel. Since HTTP communication usually takes place over TCP connections [85], the CK module intercepts TCP data flow at a location between socket and the TCP protocol implementation (Figure 35 (a)). We make this choice for the following three reasons. *First*, running CK under the system call interface can guarantee its backward-compatibility, since the interfaces between user space and kernel space remain untouched. *Second*, implementing the caching service above the TCP layer allows us to use socket information to distinguish different HTTP transactions. *Third*, running CK at a hight level in

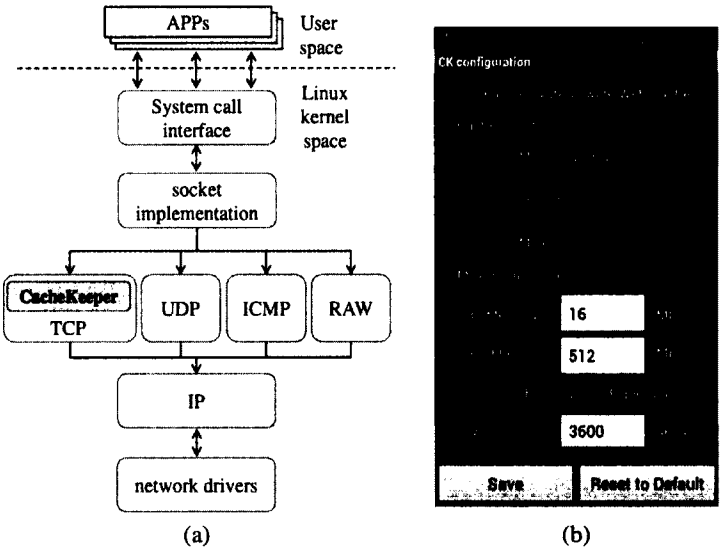


Figure 35: CacheKeeper implementation: (a) location in Linux kernel; (b) the user configuration interface.

kernel's network data flow avoids the needs of considering packet fragmentation, which lowers implementation complexity.

The HTTP transaction handler. The transaction handler inspects every intercepted TCP message, and processes those related to HTTP. Since a long HTTP response may be divided by web server into several shorter HTTP messages, the transaction handler also needs to reassemble partial HTTP response messages into a complete one. Our implementation supports reassembling for both messages with explicit `Content-Length` header field and messages using chunked transfer encoding [85]. The outstanding request table (ORT) is implemented as an array with 128 entries. According to our experience, 128 ORT entries are enough because the amount of concurrent HTTP requests waiting to be served is not a larger than 100 in all of our tests.

The Cache manager. The cache manager executes HTTP caching logic according to the RFC 2616 specification. Our current implementation supports caching with explicit expiration time, caching with validation and caching heuristic expiration. To achieve efficient cache lookup, the cache lookup table (CLT) is implemented as a dynamic hash table indexed by the `RU` field. The initial number

of CLT entries is 1024. When the CLT is 80% full, it is expanded by adding 256 empty entries. To achieve hash table indexing and also save memory space for the CLT, we place a hashed URL value, instead of the actual URL (which may be of hundreds of bytes), in the the `RU` field of an CLT entry (the same implementation applies to ORT entries). To achieve consistent web caching between reboots, we write the CLT to a file before system reboots or unloading the CK module, and read the CLT into memory right after the CK module is loaded.

Cache replacement. If adding a new HTTP transaction to a cache (volatile or non-volatile) will cause the cache's size exceeds the configured value, the cache manager deletes a cache entry from the cache. Our current implementation adopts the simplest replace policy: deleting the oldest cache entry. In the future we plan to implement different types of cache replacement algorithms, and evaluate how these the replacement algorithm can affect the performance of CK.

Private transaction declaration interface. To declare an HTTP transaction as private, an app adds a comment string "CK-Private" to the request's `User-Agent` header field. The transaction handler marks the `IP` field of the corresponding ORT entry as "True" if the comment string is found, and will never cache the response in the shared cache. Since web servers will ignore comments in HTTP headers, this approach will not affect the app's normal function. Please note that this method is used by apps to choose whether its HTTP responses can be put in a shared cache. This is different from the `Cache-Control:no-store` directive in RFC 2616, which is used by the web server to declare if a response should not by stored by any cache.

User configuration interface. We provide an interface, by utilizing the `/proc` filesystem, for users to configure CK and to obtain CK status. Figure 35 shows the screenshot of a CK configuration app. The configuration options include turning on/off CK, setting caching location, setting sizes of caches, enabling/disabling heuristic caching and setting heuristic expiration time.

4.5 Discussion

Caching HTTPS Traffic. HTTPS traffic can be cached by web clients. However, the result of our Android app caching measurement study did not contain statistics on HTTPS traffic. This is because contents of HTTPS transactions were encrypted, and could not be parsed by our analysis program. However, among the 1300 selected apps, only 10% of them generate only HTTPS traffic. The current design of CK does not support caching HTTPS traffic. One way to enable HTTPS caching is to generate a CK certificate accepted by both apps and web servers. This way, CK can decrypt and analyze through HTTPS traffic, perform caching and encrypt traffic back. We leave supporting HTTPS caching in CacheKeeper to our future work.

Privacy Considerations. Sharing web cache among applications brings privacy concerns. For example, sensitive objects of one app may be accessed by other apps. A more sophisticated case is that a malicious app could use the time difference of downloading certain HTTP objects to determine if the user has viewed certain web contents, which is similar to the timing attacks of website accesses [91]. The simplest and most effective solution is to disable shared caching for sensitive HTTP objects. To this end, CK allows an app to declare if an HTTP transaction is private to the app and should not be stored by the shared cache. However, this solution requires app modifications, and thus is not applicable to legacy applications. To better solve this problem, we are considering other solutions including randomizing cache access times [91, 92], and fingerprinting app web access patterns to detect malicious cross-app cache accessing.

Dynamic Web Content Support. The current design and implementation of CK adopt a "URL-indexed" cache, where URLs of HTTP requests are used as the keys to look up cached HTTP responses. While this solution works well as we will demonstrate later, it misses the caching opportunity for those dynamically

generated web contents, where the same HTTP object may be requested by using two different URLs. A promising way to improve this is to use content digest as the cache index key of an HTTP response. We leave this improvement to our future work.

4.6 System evaluation

We evaluated our CK implementation in a Samsung Galaxy S2 smartphone running Android 4.0.3.

4.6.1 Case evaluation: app performance gains

We selected 10 top ranked apps with imperfect web caching from Google Play (listed in left part of Table 9). All these apps were ranked top 20 in their categories. Before performing the measurement experiments, we first used the 10 apps, each for three minutes, on the Samsung Galaxy S2 smartphone, and recorded the user inputs when using the app. We instrumented CK such that it can record different statistics of through HTTP traffic, including the amount of total HTTP traffic, the amount of HTTP traffic served by the caching service and the amount of traffic with different cacheability.

HTTP traffic reduction. In this experiment, we aimed to investigate how the 10 top ranked apps can benefit from CK in terms of HTTP traffic reduction. We ran the 10 apps on the smartphone by replaying the recorded inputs from real user for every 30 minutes in one-day period. This is to simulate a user accessing an app every 30 minutes (note that the actual benefits achieved by CK depend on how often the web contents accessed by the user are updated, which is further determined by how often the user uses the app and how often the app updates its web contents, discussed later).

Table 9 presents the ratios of traffic obtained in the experiment. The third

Table 9: HTTP traffic ratios of the 10 tested apps.

Category	App Name	Served by CK	New & cacheable	Non-cacheable
News	Fox News (N1)	0.1967	0.4154	0.3879
	USA Today (N2)	0.4091	0.2075	0.3834
	AOL (N3)	0.3528	0.1710	0.4762
Shopping	Ebay (S1)	0.5654	0.1218	0.3127
	Craigslist (S2)	0.2512	0.3823	0.3665
	Target (S3)	0.6098	0.0734	0.3168
Weather	Weather.com (W1)	0.6035	0.0716	0.3249
	AWS (W2)	0.2472	0.1363	0.6165
Local&Travel	Yelp (LT)	0.5810	0.0164	0.4026
Sports	Coll. Scoreboard (SP)	0.6454	0.2161	0.1384
Overall		0.4205	0.2388	0.3407

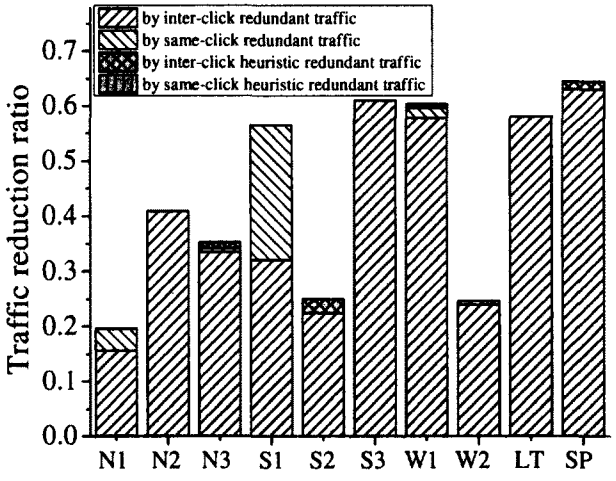


Figure 36: Source breakdown of HTTP traffic reduction ratio for the 10 tested apps.

column is the ratio of HTTP traffic served by CK, which is also the traffic reduction ratio. The fourth and the fifth column of Table 9 are the ratio of those first-time appeared cacheable traffic and the ratio of non-cacheable traffic respectively. The sum of the values in these three columns is 1. In the experiment, the overall HTTP traffic reduction ratio is 0.42. Among the 10 apps, 5 of them enjoyed a traffic reduction of over 50%. The traffic reduction ratio of an app is determined by two factors: how well web caching is implemented in the app and how often the app updates its web contents. Specifically, the worse web caching performance an

app has, the higher traffic reduction ratio it can obtain from using CK. For example, the two weather apps had a similar rate regarding content update. The weather app 1 had a higher HTTP traffic reduction ratio than the weather app 2. This is because the weather app 1 has a worse web caching performance. Meanwhile, for two apps with similar web caching performances, the app with less frequent content updates enjoys more traffic reduction. For example, the three shopping apps perform similarly in web caching: all of them do not cache image resources. In the experiment, the shopping app 1 and 3 had higher traffic reduction ratios than the shopping app 2. This is because the shopping app 2 has a much higher content update rate (it is the official app of Craigslist, which is a popular classified advertisement website where lots new listings are posted by users every hour).

Figure 36 shows the source breakdowns of the HTTP traffic reduction ratio. We can see that inter-click redundant HTTP traffic was the only major contributor to the overall traffic reduced for 9 apps. For the shopping app 1, same-click redundant traffic was another main source of traffic reduced. This suggests that it is worthwhile to pay special attention to same-click redundant traffic in CK's design.

Web content rendering speedup. We evaluated how the 10 top ranked apps can expedite web content rendering under different connection conditions by using CK. In this experiment, the smartphone was connected to the Internet via our HTTP proxy [84], which could throttle download and upload bandwidths according to user configuration. We set the transmission bandwidth at the proxy according to a recent study on 3G/4G wireless speed [93]. This study suggests that the average 3G download speeds of the four major U.S. wireless service providers range from 0.59 Mbps to 3.84 Mbps with an average value of 2 Mbps. The average 4G download speeds range from 2.81 Mpbs to 9.12 Mpbs with an average value of 6.2 Mbps. Accordingly, we chose 8 values for the download bandwidth: 0.1, 0.5, 1, 1.5, 2, 4, 8, 12, all in the unit of Mbps, and set the upload bandwidth to 1 Mbps. We ran the 10 apps, with CK enabled in the smartphone, by replaying the

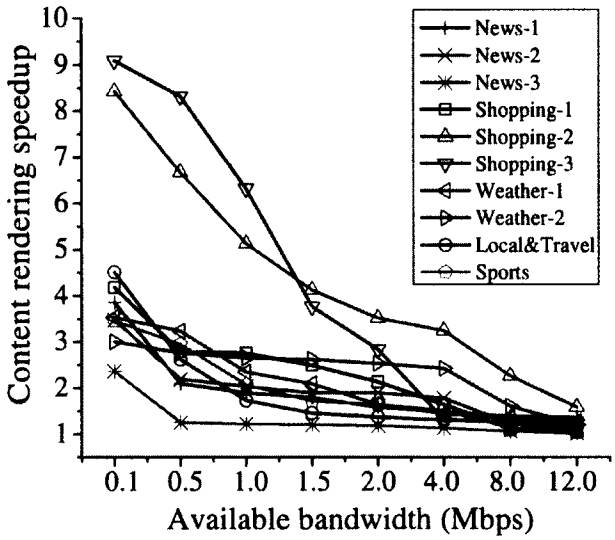
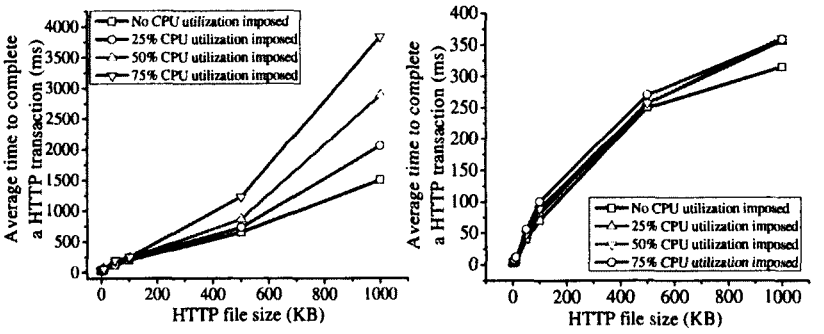


Figure 37: Web content rendering speedup of the 10 tested apps under different transmission bandwidths.

recorded user inputs under the 8 different download bandwidths for 20 rounds. In each round, we also ran the 10 apps with CK disabled. We recorded the web content rendering time for each user click, which was the time interval between the first HTTP request and the last HTTP response of all the HTTP transactions of a click. The rendering speedup was calculated as the ratio between the rendering times with and without CK running respectively.

Figure 37 shows the average content rendering speedup of the 20 rounds testing for the 10 apps. From the figure we can see that the content rendering speedup increases as the connection condition becomes worse for all the 10 apps. The shopping app 2 and 3 are more sensitive to bandwidth changes, this is because the main HTTP resources requested by these two apps are mainly large images. The average speedup of the 10 apps under the average 3G download bandwidth of the four major U.S. wireless providers (2 Mbps, reported in [93]) is 2.0. The average speedup under the average 4G download bandwidth (6.2 Mbps, reported in [93]) is around 1.5.



(a) Serving HTTP transactions from network downloading

(b) Serving HTTP transactions from local cache

Figure 38: Transaction times under different system loads.

4.6.2 Controlled evaluation

Effects of High System Load. Mobile devices usually have constraint computational resources, and thus, mobile app performances are more sensitive to system load changes than their counterparts in PCs. Here, we wanted to investigate how CK can help mobile apps to improve their resilience to high system load. We developed a mobile app that can repeatedly download specified files from our own HTTP server with a configured interval. We used this app to download files with different sizes (1, 5, 10, 50, 100, 500 and 1000, in KB) from the server. For each file size, we repeated the download for 50 times with a 100 ms interval, and calculated the average time needed as the HTTP transaction duration for the file size. During the downloads, we imposed different background workloads on CPU so that we can see how transaction durations responded to system load changes.

We first performed the experiment without running CK. In this case, every HTTP transaction was served from network downloading. Figure 38 (a) plots the relationship between file sizes and transaction durations. We can see that transaction duration of a file increases much faster as file size increases if the background system load is high. This is because to transmit a large file, HTTP servers usually divide it into small chunks and transmit them separately. For example, our HTTP server segmented a large file into 8 KB chunks for separated

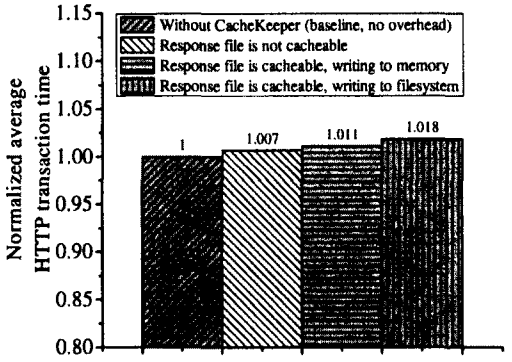
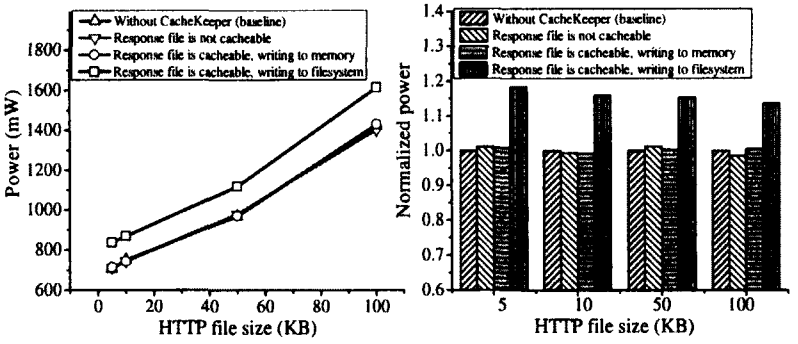


Figure 39: Processing time overhead.

transfers. Since frequent network transfers consume high CPU resource, downloading a large file needs more time under higher system load. We then performed the same experiment with CK running in the phone. In this case, except for the first download, which was served by network downloading, all the other 49 downloads were served by CK. Figure 38 (b) shows the experiment result. We can learn that when HTTP transactions were served by CK, the transaction durations were not only shorter than when served from network downloading (one magnitude less), but also more resilient to system load changes (i.e., the transaction duration for the same file size increases little as system load increases), which is helpful to offer good user experiences under high system loads. This suggests that CK is desirable in mobile devices with constraint resources.

Processing time overhead. We evaluated processing time overhead caused by CK in the case of cache miss. There are two cases for processing time overhead on cache miss. First, if the HTTP response to the cache missed request is not cacheable, processing overhead by CK comes from searching the CLT for a matched cached response. Second, if the HTTP response is cacheable, additional processing time overhead comes from caching the response. We performed the experiment by downloading a 100 KB file from our HTTP server for 50 times with a 100 ms interval. We first ran the experiment with CK disabled, and recorded the average transaction duration as the based line value. Then we ran



(a) Absolute power values for downloading files with differet sizes

(b) Normalized power values for downloading files with differet sizes

Figure 40: Power consumption overhead.

the experiment with CK enabled while enforcing the two cases of cache misses respectively, and recorded the corresponding average transaction duration. For the first case of cache miss, we configured HTTP responses as "`no-stored`". To enforce the second case of cache miss, we used different file names for each downloading. We also configured CK so that we could compare the difference of caching responses to memory and caching responses to filesystem files.

Figure 39 shows the normalized transaction durations under different scenarios. When responses to cache missed requests were non-cacheable, the processing time overhead was less than 1%. When responses are cacheable, the case of writing responses to memory had an processing time overhead of 1.1%. and the case of caching responses to files had an overhead of 1.8%. This result suggests that our implementation of CK incurs a small processing overhead on cache misses.

Energy overhead. In this experiment, we evaluated the energy overhead of CK in the case of cache miss. Similar to processing time overhead, energy overhead on cache miss also has two cases: the case that responses are not cacheable and the case that responses are cacheable. When responses are non-cacheable, the energy overhead is looking up the CLT for a matched cached response. When responses are cacheable, additional energy overhead comes from writing responses to memory and/or files. We performed the experiment by

downloading files with different sizes (5, 10, 50, 100, in KB) from our HTTP server. For each file size, we repeated the download for 50 times with a 100 ms interval. We measured phone power consumption using the Monsoon power monitor [73]. To obtain the baseline power consumption, we first ran the experiment with CK disabled. Then we ran the experiment with CK enabled while enforcing the two cases of cache miss using the same methods as in the processing time overhead experiment.

Figure 40 (a) plots the absolute power values, and Figure 40 (b) shows the normalized power values for different file sizes. From the result we can learn that CK incurs negligible energy overhead when responses to cache-missed HTTP requests are non-cacheable and when cacheable response are only written to memory. When cacheable responses are written to files, about 15% power overhead is incurred.

4.7 Conclusion

We propose and design CacheKeeper, an OS web caching service for smartphones. To motivate the work, we have performed a comprehensive measurement study on web caching functionality of 1300 top ranked Android apps. The measurement results suggest that imperfect web caching is a common and serious flaw for Android apps generating web traffic. We have implemented CacheKeeper in Linux kernel, and performed extensive evaluations on Android smartphone. Our evaluation indicates that CacheKeeper can effectively remedy the flaw of imperfect web caching for mobile apps with small overhead.

5 Achieving Accurate CPU Power Modeling for Multicore Smartphones

CPU is a major source of power consumption in smartphones. Power modeling is a key technology to understand CPU power consumption and also an important tool for power management in smartphones. However, we have found that existing CPU power models for smartphones are ill-suited for modern multicore CPUs: they can give high estimation errors and high estimation accuracy variation for different types of workloads on mainstream multicore smartphones. In this project, we tried to understand the root cause of the limitations of the existing CPU power models, and developed an accurate CPU power modeling approach that can benefit the power management service in smartphones.

5.1 Background and related work

5.1.1 Background: smartphone CPU power management

A smartphone CPU has different states: a CPU core can be either *online* or *offline* (i.e., powered down). An online CPU core can further work in either the *operating state* or an *idle state*. The operating system of a smartphone manages the states of CPU cores to reduce their total energy consumption. There are three CPU power management schemes used in modern smartphones: *CPU performance state management*, *CPU idle state management*, and *CPU hot-plugging*.

We briefly introduce each of them, with the emphasis placed on its implementation in the Android OS and the quad-core Nexus 4 smartphone.

CPU performance state management. When a CPU core works in the operating state, all processor components are powered up. In the operating state, a CPU core may operate in different *performance states* (also known as "*P-states*" in the ACPI specification [94]). Practically, each P-state is associated with a fixed CPU operating voltage and frequency. A technique called Dynamic Voltage and Frequency Scaling (DVFS) is employed to adjust the operating voltage/frequency, and thereby switch between different P-states.

The Nexus 4 smartphone supports 12 different CPU operating frequencies, ranging from 348 MHz to 1,512 MHz. Operating frequencies can be independently set in each CPU core. Choosing a proper frequency for an operating processor core is an important task for CPU P-state management. In Android kernel (Linux-based), a subsystem called "*CPUfreq*" specifically copes with this task by dynamically adjusting the operating frequency according to the system load [95].

CPU idle state management. Smartphone OS may put an online CPU core into an idle state when there is no workload. CPU idle states are called "*C-states*" in the ACPI specification [94]. CPU in different C-states have different CPU components switched to low power mode to reduce power consumption.

Table 10 shows that the Nexus 4 smartphone has four CPU idle power states: C0[1], C1, C2, and C3. A CPU core in the state C0 only disables most of the CPU clocks, while keeping the core logic powered up. A core in the state C1 has its logic powered down, but retains the in-core L0/L1 cache content by keeping the cache powered up. A core in the state C2 has more power savings than in the state C1, since the in-core L0/L1 cache are also flushed and disabled. Finally, a core in the state C3 achieves the most power savings by further disabling the

[1]In the ACPI specification, "C0" refers to the operating state, and "C1, C2, ⋯" refer to the idle states. Here we follow the naming convention in the Nexus 4 stock kernel source code, where the state C0 refers to the shallowest CPU idle state.

Table 10: CPU Idle Power States in Nexus 4.

Idle State	Name	Idle System Power (mW)	Latency (μS)†
C0	Wait for Interrupt	433	1
C1	Retention	390	415
C2	Power Collapse Standalone	330	1300
C3	Power Collapse	200	2000
Without entering idle states		1,060	0

†: The data is obtained from the Nexus 4 kernel source code.

shared L2 cache.

We have measured the idle system power of each C-state in a Nexus 4 smartphone. The third column of Table 10 shows the results. As a comparison, we have also measured the case of not entering C-states, where the idle system power is 1,060 mW. Entering a C-state can save much power when a system is idle. It also shows that power consumption of different C-states varies: the power of C0 is as much as 2.1 times of the power of C3. Consequently, entering different C-states may cause significantly different power savings. In old single-core smartphones, there are less CPU idle power states. For example, the Nexus S smartphone has only one idle state, which is equivalent to the C0 state in Nexus 4. Therefore, CPU idle states do not play a critical role in CPU power consumption on old single-core smartphones as they do on modern multicore smartphones.

Although entering idle power states reduces power consumption when a CPU is idling, it comes with a price of state switching overhead: the deeper an idle state is, the larger the switching overhead will be. The fourth column of Table 10 shows the latencies of switching between the operating state and an idle state.This operating/idle state switching latency has significant impact on performance of time-critical operations, such as video and audio decoding. In Android kernel (Linux-based), a subsystem named "*CPUidle*" is specifically designed for managing the

CPU idle states.When the OS finds no task to schedule, it directs the control to the *CPUidle* subsystem, which then decides to put CPU into a proper idle state based on several factors, including the predicted length of the current idle period (based on the information on the kernel scheduler and timers) and the operating/idle switching latency of each individual idle state.

CPU hot-plugging. In a multicore smartphone, the OS turns a CPU core offline when the CPU core has no workload for a certain period of time, and takes it back to online when the core is needed on the fly. This technique is known as CPU *hot-plugging*. While the CPU hot-plugging technique saves more power than the deepest CPU idle state, its major disadvantage is that the unplugging/re-plugging process requires expensive global operations, which causes a large amount of latency [96]. In Nexus 4, the stock Android system uses a user space daemon called "*mpdecision*" to manage the CPU hot-plugging process. The daemon monitors the load on CPU cores, and turns cores online/offline through the */sys* interface.

5.1.2 Related work

Existing approaches for modeling CPU power consumption can be classified into two categories as below.

CPU frequency/utilization based approaches. Existing approaches for modeling CPU power consumption [24,25,28,29,97] on smartphones are all CPU frequency and utilization based. They assume CPU frequency and utilization as two major factors impacting CPU power consumption. While this assumption works well for single-core smartphones, where CPU idle states have little impact on CPU power, it does not hold for multicore smartphone with multiple CPU idle states, in which power consumptions are significantly different.

Some existing approaches of CPU power modeling also consider CPU idle states [27, 28]. Specifically, Koala [27] proposes a model based approach to estimate runtime system power. In this approach, CPU idle states are considered

as a factor affecting system power consumption. However, Koala only considers the time duration of each idle state, while ignoring overheads of the operating/idle transitions. As we have showed before, even for two workloads with the same CPU frequency/utilization and the same residency of idle states, the CPU power consumption could have more than 20% difference. Moreover, it only reports evaluation results on the x86 architecture. Sesame [28] also considers CPU idle states in modeling CPU power consumption. However, it does not provide description about how this particular information is used in the modeling process. Similar to Koala, the idle states are only considered in the laptop model (x86-based) in Sesame. In our work, we focus our attention on measuring/investigating the impacts of CPU idle state on ARM-based smartphone CPUs. We also developed a new idle-state-aware CPU power modeling approach based on the investigation results.

CPU hardware events based approaches. Another way of performing CPU power modeling is to model the relationship between CPU power and CPU hardware events [98–101]. For example, Power Containers [98] considers a linear model between CPU power consumption and a series of hardware events, including retired instructions, floating point operations, last-level cache requests, and memory access. While the CPU hardware events based approaches work well for PC or server CPUs, whose ISA are mostly x86 based, they cannot be applied in current smartphones. This is because although many hardware events are recommended to be implemented in the hardware monitor by the ARMv7 architecture specification [102], only very few of them are mandated. For example, in the CPU used by the Nexus 4 smartphone, only the hardware events of instruction rate, number of instructions retired, and branches executed and missed are implemented, which is not enough to support the hardware events based modeling.

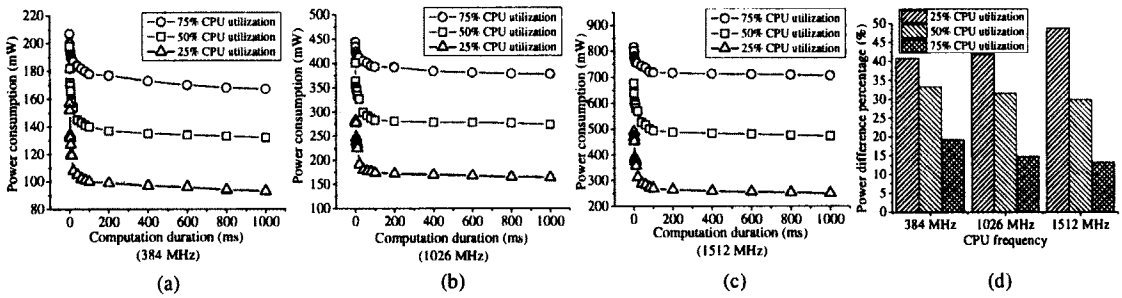


Figure 41: Workloads running in multicore CPU with the same CPU utilization and frequency consume notably different amounts of CPU power.

5.2 Limitations of the existing smartphone CPU power models

The existing power models [24–29] achieve a good accuracy (e.g., more than 90%) on single-core smartphones such as the Nexus one and Nexus S smartphones. Those models consider only CPU utilization and operating frequency as predictors in modeling [24–26]. Usually, they use a linear CPU power model: for each CPU frequency f, they estimate the power consumption of a CPU core as:

$$P_{cpu} = \beta \times U_{cpu} + c \tag{5.1}$$

where U_{cpu} is the CPU core utilization, and β and c are two constant parameters whose values are determined via linear regression during the model generation process.

However, the existing CPU power models are not suited for modern multicore CPUs. In particular, we find that CPU power consumptions in two quad-core CPU smartphones with different chipsets, Nexus 4 and Samsung Galaxy S4[2], exhibit a large range of variation even when both CPU frequency and utilization are fixed.

In our experiments, we first use a *workload generator program* that periodically

[2]Technically the Samsung Galaxy S4 smartphone has 8 CPU cores: a quad-core ARM Cortex-A7 and a quad-core ARM Cortex-A15. However, these two quad-core CPUs cannot run concurrently, since the smartphone is using the ARM big.LITTLE task migration use model [103].

performs continuous computation followed by an idle period (see Figure 1) in a Nexus 4 smartphone. By controlling the ratio of the idle period to the computation period, the workload generator program generates workloads with different CPU utilizations. In the continuous computation, the program runs a busy loop of computing a large prime. By changing the busy loop count, we can also control the length of each continuous computation period. We find that for a fixed CPU frequency, when we adjust the length of the continuous computation while fixing CPU utilization (by adjusting the length of idle period accordingly), the power consumption of a CPU core exhibits a large range of variation. For example, Figure 41(a) shows the power consumption of a CPU core[3] of the Nexus 4 smartphone when the operating frequency is fixed at 384 MHz. With fixed CPU utilization, the power consumption of the CPU core drops while the duration of the continuous computation increases. Figure 41(b) and Figure 41(c) show the results when CPU frequency is 1,026 MHz and 1,512 MHz, respectively. They show exactly the same trend. Figure 41(d) further summarizes the difference of power consumption with the three CPU frequencies. Each value in Figure 41(d) is the percentage of the difference between the maximum and minimal powers over the maximum power for each frequency/utilization configuration. It shows that when CPU frequency and CPU utilization are fixed, the CPU consumption difference between different workloads is significant, especially when the CPU utilization is at a low level. For example, when frequency/utilization is fixed at 1,512 MHz/25%, the power difference can reach as high as 50%. As we will explain later, this is because the less a CPU core is being utilized, the more chance the *CPUIdle* subsystem puts the CPU core into a deeper idle state.

The above results suggest that using only CPU operating frequency and utilization is not enough to build an accurate CPU power model for multicore smart-

[3]The CPU power consumption is measured as the system power when the smartphone is configured in a way that CPU is the only main source of power consumption. See the evaluation section for more details.

Table 11: Time duration per second and number of state entries per second in two workloads of the same CPU utilization (50%) under the same CPU operating frequency (1,512 Mhz).

Idle State	Time duration (ms)		# of state entries	
	W1	W2	W1	W2
C0	491.85	1.08	468	1.99
C1	0	0	0	0
C2	1.18	1.43	0.1	0.2
C3	5.12	496.86	0.2	7.3

phones. As we introduced previously, in modern multicore smartphones like Nexus 4, the CPU power is determined not only by the CPU frequency and utilization, but also by the CPU *idle power states*, which are not considered in the existing smartphone CPU power models. Modern multicore CPUs like the one of Nexus 4 have multiple idle power states which have significantly different power consumptions. When utilization is fixed, prolonging the duration of continuous computation causes the corresponding idle period to increase accordingly. Longer idle period allows the OS to put the CPU core into deeper idle states more frequently, which in turn lowers the CPU power consumption.

To further demonstrate how CPU idle power states can affect power consumptions of different workloads running with the same CPU frequency/utilization, we list in Table 11 the statistics of the idle states of two workloads (W1 and W2) that were run in a Nexus 4 smartphone: the time duration per second of each state, and the total number of entries per second of each state. These two workloads were run with the same CPU frequency (1,512 MHz) and the same CPU utilization (50%), but they had significantly different power consumptions (644 mW for W1, and 499 mW for W2). The two workloads had notably different idle state transition statistics as shown in Table 11: with the workload W2, the CPU core stayed at the deepest idle state much longer than with the workload W1. This explains why W2 consumed significantly less CPU power than W1. Note that because the stock

Nexus 4 kernel does not enable the idle state C1, the numbers of C1 in Table 11 are 0s.

We have also performed the experiments in a Samsung Galaxy S4 smartphone, which is equipped with a chipset different from Nexus 4, and obtained similar observations. With the Galaxy S4 smartphone, when the CPU frequency/utilization are fixed at the top frequency/25%, the power consumption of a CPU core could exhibit up to 38% difference when we adjust the length of continuous computation in the workload generator program. The power difference we observe in Nexus 4 (50%) is slight higher than in Galaxy S4 (38%). This is because Nexus 4 implements deeper idle power states than Galaxy S4 does. As an evidence, according to our measurement, the ratio of the power consumptions of the deepest idle state over the shallowest idle state in Nexus 4 is smaller than that in Galaxy S4 (0.46 for Nexus 4, 0.51 for Galaxy S4). Since implementing deeper idle power states is a clear trend in future multicore smartphones (for more energy efficiency), we expect the possible power difference under the same CPU frequency/utilization setting will keep growing in future smartphones, which urges the need for developing a new CPU power modeling method that considers CPU idle power states.

5.3 Idle-state-aware CPU power model

In this section, we first present the development of our power modeling for the single-core case. Then, we show how the single-core power model can be extended to the multicore case. All the experiments described in this section are performed in a Nexus 4 smartphone.

5.3.1 Power modeling for a single CPU core

Similar to existing work, we use regression-based method to integrate the predictors. To determine what statistic of CPU idle states should be used as a predictor

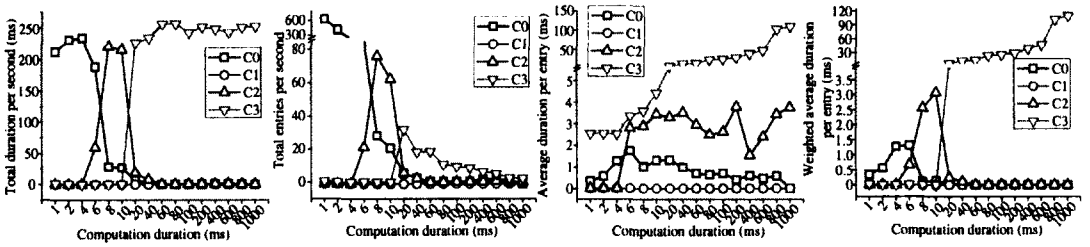


Figure 42: Single-core power model development. Figures (a)-(d) show T_{C_i}, E_{C_i}, ED_{C_i}, and WED_{C_i} for the four CPU idle states C_0 - C_3, respectively (with CPU frequency $f = 1,512$ MHz, utilization $U = 75\%$).

variable of the regression model, we first consider T_{C_i}, which is the *total time duration* that a CPU core stays in the idle state C_i per second when frequency f and utilization U are fixed. Suppose the total CPU idle time per second is T_{idle}, we have

$$T_{idle} = \sum_i T_{C_i} \tag{5.2}$$

Figure 42(a) shows T_{C_i} for idle states C_0 to C_3 when we ran our workload generator program on a single CPU core (with $f = 1,512$ MHz, $U = 75\%$). Since the stock Nexus 4 kernel does not enable the idle state C_1, statistics for C_1 remain zero in Figure 42. The figure shows that the CPU core spent more time staying in deeper idle states as duration of the continuous computation increased, because the idle period also increased accordingly. However, T_{C_i} is not a good predictor of CPU power consumption. For example, after the computation duration increased to 20 millisecond, T_{C_i} ($i = 0, 1, 2, 3$) stayed stable, but the CPU power actually kept decreasing as the the computation duration increased (see Figure 41(c)). In fact, in our experiment, the power difference could reach 24% for the same T_{C_i} ($i = 0, 1, 2, 3$) (when $f = 1,512$ MHz, U=25%).

Figure 42(b) shows E_{C_i}, which is the *number of entries* for idle state C_i per second, in the same experiment. For the same T_{C_i}, smaller E_{C_i} means less operating/idle transition energy overhead, and thus more energy savings. This explains our previous observation that CPU power kept decreasing when T_{C_i} is unchanged.

However, E_{C_i} alone is also not a good predictor of CPU power consumption, as it has no direct link to energy savings by idle states.

We then look at the *average entry duration* for idle state C_i, which is notated as ED_{C_i}:

$$ED_{C_i} = \frac{T_{C_i}}{E_{C_i}} \tag{5.3}$$

Generally, ED_{C_i} is a good predictor of CPU power, as it involves both idle state duration and state transition overhead. However, ED_{C_i} could suffer from noise, which comes from those sporadic entries of idle state C_j when the CPU enters state C_i most of the time. For example, Figure 42(c) shows ED_{C_i} in the experiment. We can see that ED_{C_3} was greater than ED_{C_0} when C_0 is the dominant idle state.

To eliminate noises in ED_{C_i}, we apply a weight w_i, which is the portion of time the CPU stay at the state C_i over the whole idle period, to ED_{C_i} to form *weighted average entry duration* WED_{C_i}:

$$WED_{C_i} = w_i \times ED_{C_i},\ where\ \ w_i = \frac{T_{C_i}}{T_{idle}} \tag{5.4}$$

Figure 42(d) shows WED_{C_i} in the experiment.

Finally, we model power consumption of a single CPU core working at frequency f as

$$P_{core} = \sum_i \beta_{C_i} \cdot WED_{C_i} + \beta_U \cdot U + c \tag{5.5}$$

where β_{C_i} and β_U are the coefficients of WED_{C_i} and the utilization U, and c is a constant. For each CPU frequency f supported by Nexus 4, we obtain the coefficients and the constant by running linear regression analysis on the training data containing different T_{C_i} and U, and the corresponding P_{core} (see the system design and implementation later).

Table 12: CPU power with different number of cores running (with utilization U=50%).

N_c	f=384 MHz			f=1512 MHz		
	P_{BL,N_c} (mW)	P_{CPU} (mW)	$P_{\Delta,core}$ (mW)	P_{BL,N_c} (mW)	P_{CPU} (mW)	$P_{\Delta,core}$ (mW)
1	62	144	82	62	495	433
2	73	213	70	73	902	415
3	73	282	70	73	1,312	413
4	73	348	69	73	1,732	415

N_c: number of cores that ran the workload.
P_{BL,N_c}: baseline CPU power with N_c cores enabled.
P_{CPU}: whole CPU power.
$P_{\Delta,core}$: power increment per core.

5.3.2 Power modeling for multicore CPU

We further conduct an experiment to study how the single-core CPU power model can be extended to multicore scenario. In the experiment, we enabled different number of CPU cores, which are running at the same frequency, and then generated the same amount of workload on each enabled core. We measure the CPU power while varying the core frequencies and utilization. Table 12 presents the results for the cases when core frequencies are fixed at 384 MHz and 1,512 MHz, and the core utilization is 50%. In the table, the power increment per core is calculated as $P_{\Delta,core} = \frac{P_{CPU} - P_{BL,N_c}}{N_c}$, where N_c is the number of cores enabled, P_{BL,N_c} is the baseline CPU power when N_c cores are enabled, and P_{CPU} is the whole CPU power measured. We can see that $P_{\Delta,core}$ is consistent for the same "frequency/utilization" with more than one core enabled, but is notably smaller than the value when there is only one core running the workload. The reason is that in Nexus 4, when there are more than one core running, the deepest CPU idle state each running core can enter is state C_2. The state C_3, where the shared L2 cache is disabled, can only be entered by core-0 when no other core is online. Therefore, $P_{\Delta,core}$ for the single-core case is always greater than that for the multicore

case.

Based on our observation, we model a multi-core CPU power consumption P_{CPU} as

$$P_{CPU} = P_{BL,N_c} + \sum_{i}^{N_c} P_{\Delta,core,U_i,f_i} \tag{5.6}$$

where N_c is the number of cores enabled, P_{BL,N_c} is the baseline CPU power with N_c enabled cores, and $P_{\Delta,core,U_i,f_i}$ is power increment of core-i when it is working at frequency f_i with utilization U_i. For each frequency f_i, $P_{\Delta,core,U,f_i}$ can be predicted using the single-core power model developed previously, while P_{BL,N_c} is a constant value that can be measured beforehand. For Nexus 4, we need to model $P_{\Delta,core_U,f_i}$ separately for the case when there is only one core is online and when there are multiple cores are online, because these two cases have different sets of CPU idle states.

5.4 System design and implementation

We have designed and implemented a prototype CPU power estimation system using our idle-state-aware CPU model on Android platform. Figure 43 shows an overview of the system. The system contains two parts: one runs in the kernel space, and the other runs in the user space. In the kernel space, the *data collector* component collects necessary CPU usage data including the CPU frequency, CPU utilization, and CPU idle state statistics. In the user space, the *controller* component controls the procedure of model generation. To generate a CPU power model, the controller runs a set of training programs, starts the data collector, and collects CPU usage data. At the same time, we measure the CPU power consumption using a power meter. Using the measured power data and the collected CPU usage data, the *model generator* component creates a CPU power model through linear regression. Although our implementation is based on Android platform, we expect the system design can also work on other mobile

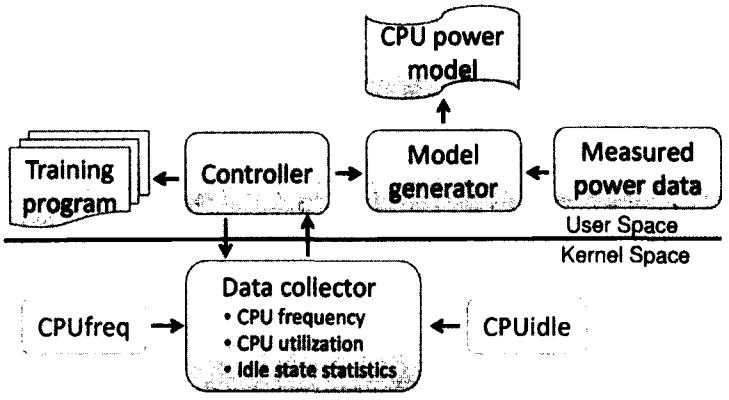


Figure 43: System overview.

platforms such as Windows Phone and iOS.

Collecting data in the kernel. We design a data collector to work in the kernel space for lightweight and efficient data collection. A design alternative is to periodically sample CPU utilization and CPU idle states in the user space via the high-latency */proc* and */sys* filesystems. However, because our power model needs CPU statistics for each working frequency, which may change tens of times per second, the user space alternative would need to poll the kernel with an equally high frequency, which is impractical and inefficient. With our kernel-mode data collection approach, we can aggregate raw data, and report only the aggregated data to the user programs via the system call interface. Consequently, we significantly reduce the number of user-kernel mode switching, and thus introduce much less system overheads in collecting the data. Moreover, running the data collection in the kernel allows us to obtain fresh and accurate data without the latency of user-kernel mode switching.

To guarantee accuracy, it is straightforward to periodically sample data in the kernel, with the sampling rate set to the highest possible value of frequency changing rate. However, this method would incur unnecessary system overheads, since it requires a high sampling rate even when the actual CPU frequency changing rate is low. We take a different method in our implementation. We take advantage of the *CPUfreq* and *CPUidle* subsystems of the Android kernel to collect data

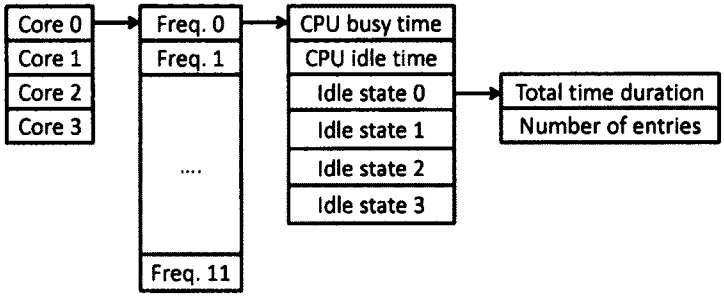


Figure 44: Data structure used in the data collector.

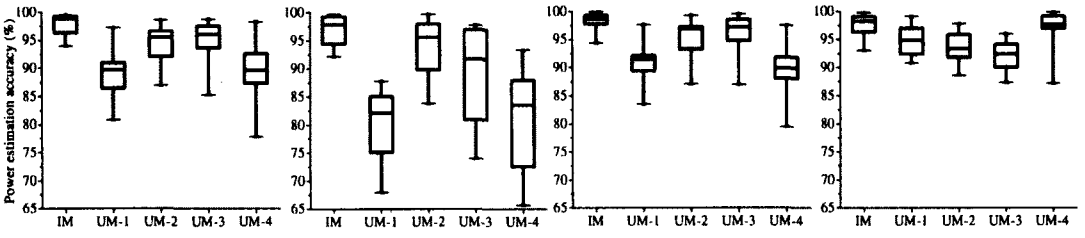
efficiently. Specifically, we piggyback our data collecting with activities of the sub-systems. We have instrumented the subsystems so that we know when the CPU frequency or CPU idle state are changed. Each frequency change in *CPUfreq* triggers a new process of data collection for the new working frequency. For each CPU idle state change in *CPUidle*, we collect new data about the previous CPU idle state and aggregate them to the existing data. Therefore, our data collection automatically adapts to CPU frequency changes, and thus avoids unnecessary system overheads.

Figure 44 shows the data structure used in our data collector (assuming a Nexus 4 smartphone is used). We collect the CPU usage data for each CPU core and each CPU frequency separately. Each CPU core has an array of 12 CPU frequencies. For each CPU frequency, we record the total CPU busy time and the total CPU idle time, based on which the CPU utilization can be calculated. We also record the CPU idle state information, including the total residency time duration and the total number of entries of each CPU idle state. With the data structure in Figure 44, we do not need to record the raw data (e.g., the CPU usage data of every trigger of data collection). Instead, for each trigger of data collection, we simply update the corresponding values in the data structure to aggregate the new data with the existing data. As a result, the data collector consumes a small fixed amount of memory, which is independent of the time duration of data collecting. Compared to recording the raw data, this approach also uses much less memory,

especially when the data collecting time is long.

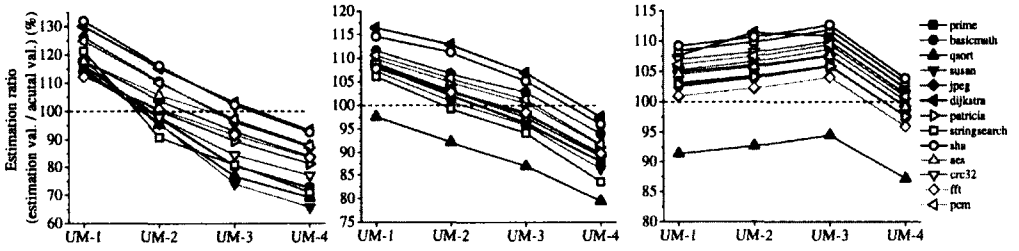
Generating CPU power model. To generate a CPU power model, we have run a set of training programs with various workloads and CPU usage patterns. We use the workload generator described previously to create training programs with various CPU frequencies, utilization, and various continuous computation durations. For each CPU frequency, we train 3 CPU utilization levels (25%, 50%, and 75%). For each CPU utilization level, we train 8 computation durations (1 ms, 2 ms, 4 ms, 8 ms, 20 ms, 40 ms, 80 ms, and 200 ms). For each CPU frequency, we also train the CPU idle case (5% utilization), and the CPU busy case (100% utilization), but with a fixed computation duration (100 ms). In total we have created 312 different training programs. We first enable only one CPU core, and run these training programs on the CPU core to generate the single-core power model. Then we enable all cores, and run the training programs with an identical process on each core, to generate the multicore power model. The whole model generation procedure takes about 2 hours. It is worth noting that the ground-truth CPU power consumption is obtained manually by using power meter. One could also obtain the ground-truth value by referring to the battery interface [24, 28], which allows for automated model generation. We opted to manual measurement because we wanted to reduce the possible errors introduced by using the battery interface.

Applying CPU power model. We have written a user space CPU power estimation C library that supports our CPU model in user space programs. The library gets CPU statistics from the data collector located in the kernel as shown in Figure 43, calculates the estimated CPU power consumption, and reports information to user programs as requested. The interfaces provided by our C library to user programs include starting and stopping the CPU power estimation period, getting the estimated CPU power consumption of the estimation period, and getting different CPU statistics, such as CPU online information, CPU utilization, and CPU idle



(a) random CPU utilization (b) 30% CPU utilization (c) 60% CPU utilization (d) 90% CPU utilization

Figure 45: Single-core model accuracy with MiBench benchmarks.



(a) 30% CPU utilization (b) 60% CPU utilization (c) 90% CPU utilization

Figure 46: Estimation ratios of the four utilization based models (single-core).

states information.

In total, our implementation has about 3,000 lines of code (LOC) in C programming language, with 1,300 LOC in kernel implementation and instrumentation, 800 LOC in the controller component, 500 LOC in the CPU power estimation C library, and 300 LOC in the model generator component.

5.5 Evaluation

5.5.1 Experimental Setup

We used a Nexus 4 smartphone, which has a 1.5 GHz quad-core Qualcomm Snapdragon S4 Pro CPU, and runs Android 4.2. The Qualcomm Snapdragon S4 Pro is a representative design of symmetric smartphone multicore CPUs. It has been widely used on many mainstream multicore smartphones from various of manufacturers, such as Google Nexus 4, HTC Droid DNA, LG Optimus G, Sony Xperia Z, and Samsung Galaxy S4 AT&T version. Our CPU power model

Table 13: Benchmarks tested in the evaluation.

Benchmark	Description
prime	Compute a large prime.
basicmath	Perform simple mathematical tasks.
qsort	Quick sort over an array of strings.
susan	Susan image recognition.
jpeg	Encode/decode a JPEG image.
dijkstra	The shortest path Dijkstra algorithm.
patricia	Patricia trees of routing tables.
stringsearch	Search for given words in phrases.
sha	SHA secure hash algorithm.
aes	Advanced Encryption Standard (AES).
crc32	32-bit Cyclic Redundancy Check (CRC).
fft	Fast Fourier Transform (FFT).
pcm	Pulse Cod Modulation (PCM).

should also work for these smartphones. We measured the system power consumption using a Monsoon power meter [73]. Since we focus on the CPU power consumption, we disabled other hardware components as much as possible including turning off the screen, network interfaces (cellular, WiFi, Bluetooth, and NFC), and sensors (GPS, accelerometer etc.). We also killed all the background services and processes that were not necessary. Note that the measured CPU power (i.e., the ground truth value) include power consumption by CPU, memory, and flash disk. Since our training programs also include memory activities, we expect power consumption on flash disk will incur small impact on the accuracy of our CPU models. Each experiment was repeated for 5 times and we report the average results.

Benchmarks. We used 13 benchmark programs from MiBench, which is a free and commercially representative embedded benchmark suite [104]. As shown in Table 13, these benchmarks cover a diverse set of computation types that are widely used in networking, security, telecommunication, image process-

ing, and many other scenarios and applications. We used our workload generator described previously, which periodically performs continuous benchmark computation followed by an idle period, to generate benchmark workloads with different CPU utilizations. To generate workloads with random CPU utilization, we randomly chose the length for *each* continuous computation and idle period. Depending on the computation type, the continuous computation periods ranged from 10 ms to 1000 ms, 250 ms on average.

Real applications. Besides the above benchmarks, we also used the following 5 applications to evaluate our CPU power model.

- *Web browsing*: we used the Dolphin Browser [105] to load five web pages pre-downloaded from `www.nytimes.com`. The five pages include the homepage and four subpages. Dolphin Browser is a popular web browser similar to Google's Chrome browser, both of which are based on the WebKit engine. We chose the Dolphin browser because it provides more control interfaces, which allow for automated tests.

- *Map*: we used Google Map to browse an offline map with operations including zooming in/out, swiping, and moving the map. We used the tool [19] to capture and replay the user inputs on the touch screen, so that we could operate on the map with desired operations automatically.

- *App loading*: we launched 8 real apps including Kingsoft Office, ThinkFree Office, Chrome browser, Firefox browser, Opera browser, Google Map, Baidu Map, and Ezpdf reader. We did not choose any games because (1) the loading processes of many CPU intensive games (e.g., Angry Birds) terminate when the screen is turned off, and (2) these games usually use GPU for graphic processing, but GPU is not considered in our power model.

- *Video decoding*: we used Dolphin Player to play a MP4 video clip (30 frames/sec, 611 kbps bitrate) for 20 seconds. We configured Dolphin Player to do video

decoding in software using CPU rather than the dedicated video decoding hardware.

- *Audio decoding*: we used Google Music to play a MP3 song clip (44.1 KHz sample rate, 64 kbps bitrate) for 20 seconds. The Google Music decodes audio file with software.

Please note that the goal of conducting experiments on real applications is to evaluate how our power modeling approach, which focuses on estimating power consumption of the CPU component, works on real app workloads in addition to those ported from MiBench. If one wants to estimate the power consumption caused by a particular app, she also needs to consider power consumption generated by other hardware components (e.g., WiFi, Bluetooth) [24].

CPU power models to compare. To compare our idle-state-aware CPU power model (labeled as IM) with existing CPU power models, we generated 4 utilization based CPU power models (i.e., traditional CPU power models that consider only CPU frequency and utilization) as follows. We used the same training programs as in our model generation process, but only considered CPU frequency and utilization, ignoring the CPU idle states. The 4 utilization based models (labeled as UM-1, UM-2, UM-3 and UM-4) were generated using 4 different computation durations: 2 ms, 8 ms, 20 ms, and 200 ms, respectively. Once we generated the single-core power models, we further created the corresponding multicore models according to the procedure described previously. It is worth noting that in previous work, utilization based CPU power models were trained only on single-core CPUs. For fair comparison, we extended the CPU utilization based power models to multicore CPU case using the same method we used in our CPU idle state based power model.

We define the accuracy of a power model as follows:

$$Accuracy = 100\% - \frac{|P_e - P_m|}{P_m}\% \tag{5.7}$$

where P_e is the power estimated by the power model, and P_m is the power measured using the power meter.

5.5.2 Experimental Results

We evaluated our prototype system from two aspects: accuracy of our CPU power models and system overheads.

Accuracy of single-core models

We first evaluated the model accuracy when only a single CPU core was used.

Benchmark experiments results. Figure 45 (a) shows the accuracy of single-core models with the 13 MiBench benchmarks programs when CPU utilization was randomly decided in the way described previously. In the figure, the bar in a box is the average accuracy of the model. The upper and bottom borders of a box represent 75 percentile and 25 percentile. The tips of the upper and bottom whiskers represent the max and min values. On average, our model achieved a high accuracy of 98%, with a small variation ranging from 94% to 100% for different benchmarks. The average accuracy and the range of accuracy variation of the four utilization based models were (with the variation range shown in the parenthesis): 89% (81%-97%), 94% (87%-99%), 95% (85%-99%), and 89% (78%-98%). We can see that our model significantly outperforms the utilization based models in terms of *estimation accuracy* and *accuracy stability*. Although the average accuracy of UM-2 and UM-3 were not far below that of our model, they exhibited a much larger range of accuracy variation for different benchmarks. This is *because different benchmarks have different CPU usage patterns, which further causes different patterns of CPU idle state entries. The utilization based models were unable to capture the effect of these CPU idles state changes, which are important dynamics affecting CPU power consumption. On the contrary, our model can well cope with this dynamic usage pattern, since it is designed with the*

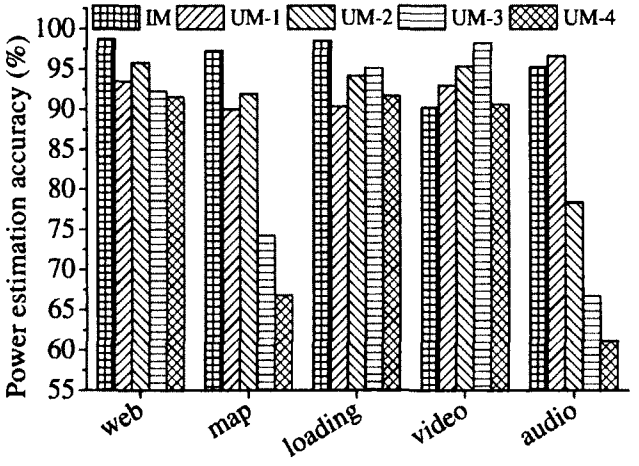


Figure 47: Single-core model accuracy with real mobile applications.

impacts of idle states in mind.

The accuracy of the existing utilization based models are also subject to CPU utilization. Figure 45 (b), (c) and (d) show more results when the CPU utilization was fixed at 30%, 60%, and 90%. We can see that the utilization based models gave notably high errors in some cases, especially when CPU utilization was at a low value. For example, when the CPU utilization was 30%, the accuracy of model UM-4 was only 66% in the *susan* benchmark, and the accuracy of model UM-1 was only 68% in the *sha* benchmark. This is *because when CPU utilization was low, there were more idle time, which in turned led to more dynamic pattern of idle state entries.*

To further study how different types of workloads and different CPU utilizations could affect the existing utilization based models, we show in Figure 46 the *power estimation ratio* of UM-1 to UM-4 when testing the 13 benchmarks with CPU utilization fixed at 30%, 60%, and 90%, respectively. The *power estimation ratio* is the percentage of the estimated power value over the measured (i.e., ground truth) power value. Thus, the closer to 100%, the better is the power estimation ratio. Figure 46 shows that for a given benchmark at fixed CPU utilization, it is possible to find a CPU utilization based model to achieve a high estimation accuracy. However, that model would have a much lower model estimation accuracy

in some other benchmarks and other CPU utilization levels. For example, when the CPU utilization is 60% (Figure 46(b)), UM-2 achieves almost 100% estimation ratio for benchmark *stringsearch*, but UM-2 would estimate about 15% more than the ground truth value if it is used for benchmark *dijkstra*. Another example is that UM-3 achieves an estimation ratio slightly more than 95% for benchmark *susan* when CPU utilization is 60% (Figure 46(b)). However, the estimation ratio for the same benchmark drops below 75% when CPU utilization is 30% (Figure 46(a)). In sum, *it is not possible to have a single CPU utilization based model to achieve a high and consistent modeling accuracy in all the benchmarks and CPU utilization levels. On the contrary, our model, which considers CPU idle states and thus can adapt to variation of CPU usage pattern, is able to achieve a consistently high estimation accuracy in all the benchmarks and different CPU utilizations.*

Real application experiments results. The similar observations can be found in the real application experiments as well. Figure 47 shows the single-core model accuracy in the five real application experiments. We can see that our model also achieved a high accuracy, 96% on average, with a variation ranging from 90% to 99% for different applications. The accuracy is slightly lower than that of the benchmark experiment. This is likely because the applications had more flash disk operations, but our model does not consider flash disk. Our model had the lowest accuracy of 90% in video decoding. This is probably because that the player used GPU which is also not considered in our model. For the utilization based models, their accuracy in the real application experiments exhibited a large range of variation. The average accuracy and the range of accuracy variation were: 93% (90%-97%), 91% (78%-96%), 85% (67%-98%), and 80% (61%-92%).

We also examined the relationship between power estimation ratio and CPU utilization for the real application experiments. Figure 48 shows the estimation ratios of all the models when the CPU utilization was different in the Web browsing application. We controlled the CPU utilization by changing the time interval be-

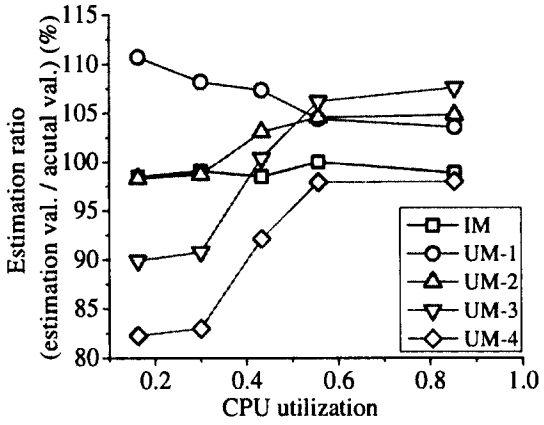


Figure 48: Estimation ratio vs. utilization.

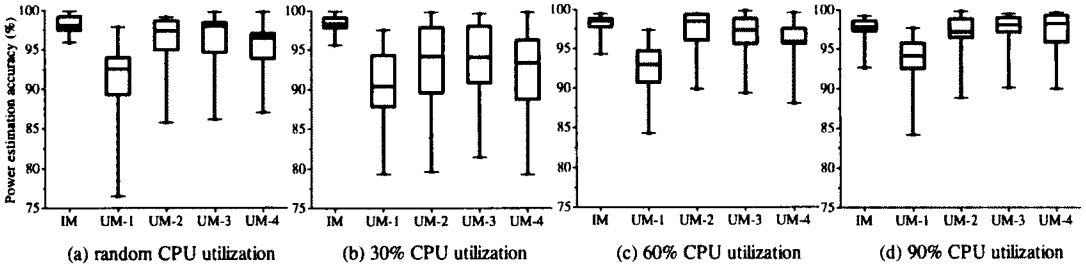


Figure 49: Multicore model accuracy with MiBench benchmarks.

tween loading the webpages. We can see that the CPU utilization based models gave a large range of accuracy variation when the CPU utilization was different. In particular, when the CPU utilization was low, they gave a lower model accuracy, which was also observed in other applications. The curve of our model is much flatter and the estimation ratios are consistently close to 100%, indicating that our model is also able to adapt to CPU utilization changes and achieve consistent high estimation accuracy under different CPU utilizations.

Accuracy of multicore models

Figure 49 (a) shows the multicore model accuracy results when we ran the benchmarks and applications with randomly decided CPU utilization using all the four CPU cores. Figure 49 (b), (c) and (d) show the result when CPU utilization was fixed at 30%, 60%, and 90% respectively. Figure 50 show the result for the real ap-

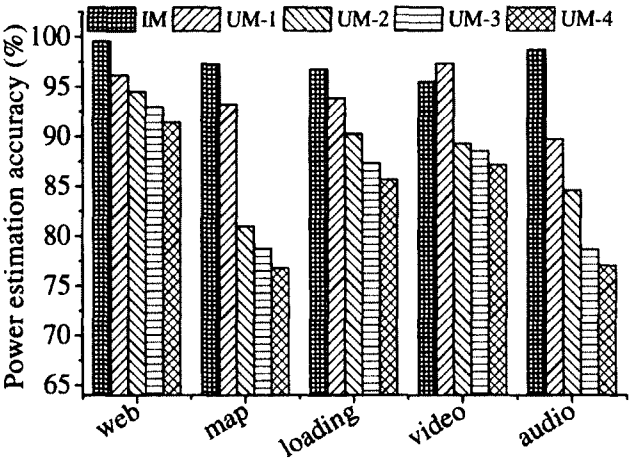


Figure 50: Multicore model accuracy with real mobile applications.

plication experiments. We observed similar results: our model achieved a higher average accuracy, and a much smaller range of accuracy variation than the existing utilization based models.

Compared to the single-core case, the accuracies of the four CPU utilization based models are relatively higher, and the differences among the four models are relatively smaller. This is because the Nexus 4 smartphone allows only two CPU idle states (C0 and C2) when multiple CPU cores are enabled. Thus, the impact of CPU idle states become smaller. However, we still have the same observations as in the single-core case: 1) our model has a consistently high accuracy in all the benchmarks and applications, and significantly outperforms the CPU utilization based models; 2) the CPU utilization based models have a large range of model accuracies in different benchmarks and application, and give a lower accuracy when the CPU utilization is lower. *As smartphone CPUs are becoming increasingly powerful, smartphone CPU utilization is usually low for the most of time. Thus, the CPU utilization based models tend to generates high errors in practice. On the contrary, our idle-state-aware CPU power model is able to adapt to CPU usage pattern changes and utilization changes, and thus can accurately estimate CPU power consumption with different workloads and different CPU utilizations.*

System overheads

From the 312 training programs, we chose those that cause the most frequent frequency changes and idle state entries to evaluate the CPU overhead of our system. On average, the chosen workloads incur about 40 frequency changes per second and about 450 entries of CPU idle states. Although our implementation should have the maximum system overhead when running these workloads, we have seen no noticeable CPU usage increase. This is because our data recording and reporting process is extremely lightweight: only several variable updates when a frequency change or idle state entry happens, and the data are reported to user space only at the beginning and end of the power estimation period. As for the memory usage, our prototype implementation use about 8 KB kernel memory, with the majority consume by the data recording data structure.

5.6 Conclusion

We demonstrated that existing CPU utilization based power models are ill-suited for modern multicore smartphones. Without considering the impacts of CPU idle states, existing power models give high errors in multicore smartphones. To address the limitations of existing power models, we developed an *idle-state-aware* CPU power model for accurate CPU power modeling in multicore smartphones.

We have designed and implemented a prototype system of our new CPU power modeling approach using the quad-core CPU Nexus 4 smartphones, and also conducted comprehensive evaluations using a diverse set of benchmarks and real applications. Experimental results show that our CPU power model achieves a high model accuracy, which significantly outperforms the existing CPU utilization based power models, with negligible system overheads.

6 VProof: Lightweight and Privacy Preserving Vehicle Location Proofs

Location based service (LBS) application is a new and popular category of smartphone applications. A proper location proof scheme is critical to providing trustworthy location based services to smartphone users. In this project, we designed a lightweight and privacy-preserving location proof scheme for LBS applications in smartphones.

6.1 Background and related work

6.1.1 Background

Location based services is a major category of services provided by mobile applications. For example, in Intelligent Transportation Systems (ITS), popular category of applications is that vehicles report information about the transportation system elements (e.g., drivers and road conditions) to the ITS system for services like real time traffic control and roads maintenance [31,32]. Location proofs allow ITS operators to verify the validity of reports submitted by vehicles, and thus are important for deployment of such data collection applications. Otherwise a malicious user (i.e., vehicle, we will use "user" and "vehicle" alternately) can launch an attack to the ITS system by reporting fake information about places where he did not actually visit. The damages of the attack are particularly serious, since

the attacker can report fake information about numerous places by just clicking mouse at home.

To verify whether a vehicle's location claims match its actual historical locations, ITS operators need a location proof scheme featuring the following properties. *First*, the location proof should be lightweight. This property is extremely important in vehicular environments, since location proof issuers may need to issue location proofs to tens or hundreds of vehicles on a busy road at the same time. *Second*, the location proof needs to well preserve users' location privacy. Concerns about users' location privacy have become major considerations when deploying location-related services in ITS systems [106]. Car owners can simply opt out of providing *any* data if their privacy is threatened. *Third*, the location proof scheme needs to be able to generate fine-grained location proofs, because the locations reported in the user collected information (e.g., there is a pothole somewhere on the road) have fine granularity.

To detect malicious users who report bogus data, the conventional solution is to assign each vehicle with some cryptographic keys. Each vehicle will sign each piece of data with its secret key before uploading to the ITS system. The idea is that by having a means to track back users, the amount of bogus data will be reduced, since malicious users do not want to be caught. Similar schemes have also been proposed to protect the privacy of honest users to encourage participation [107–109]. Nonetheless, these solutions all require deploying a large scale PKI scheme to associate specific keys with individual vehicles. While this may be possible in theory, for instance, a PKI administered by a local DMV, it is less clear if this will be done in practice, especially in a large country. We can point to the difficulties in getting the different states in the U.S. to standardize on a common driver's license as evidence of the impracticality of a widely deployed PKI solution. Thus, a solution that does *not* rely on such large scale infrastructure to provide privacy protections is needed.

The idea of location proofs has been considered by other types of applications before. The general approach is to let certain authorized entities with fixed geolocations perform as location proof issuers. The location proof issuers issue location proofs, which are unique and unforgeable, to nearby location provers, who need to prove their historical locations to a location verifier later. A location prover is believed to be in the vicinity of a proof issuer at a certain time if the prover possesses valid location proofs [110–112]. We cannot apply the same approach towards vehicular environments because the location proof granularity achieved by the existing location proof solutions is coarse: they can only prove that at a certain time a user was within the communication radius of a proof issuer but not at a finer granularity. This allows a malicious user to *statically* collect the location proofs issued by a proof issuer and report fake information about places where he never visited but are within the proof issuer's communication radius. Meanwhile, the existing solutions require a proof issuer to perform multiple rounds of interactions with a prover to issue a location proof specifically to the prover, which is not scalable to vehicular environments, where lots of vehicles may be requesting location proofs at the same time.

6.1.2 Threat model

We consider the threat that malicious users target at disrupting ITS systems by reporting fake information about numerous places where they did not actually visit[1]. If there is no scheme to allow ITS operators to verify whether the reporting users have actually visited the places indicated in the reported data, a malicious user can easily generate and report bogus data about lots of places without actually visiting those places. The amount of the bogus data could overwhelm that of the honest data. Existing works for filtering abnormal data in vehicular networks [113, 114]

[1]We do not consider the threat that a malicious user physically presents at a place and report fake information about it, as we deem this threat has much less impacts than the one we are considering.

do not work in this case, because they hold an assumption that the amount of abnormal data should not be more than that of normal data. Meanwhile, as we discussed previously, we prefer not to use PKI to solve the problem. Thus, we ask the question: without using PKI systems, can we provide a scheme to let ITS operators verify if a user's historical locations are in accordance with the data he submits so that they can prevent the threat we just described?

We make the following assumptions about malicious users. *First*, malicious users have the same equipments as honest users, and have certain knowledge of the information about the RSUs in the ITS system, such as the ESSIDs and GPS locations of the RSUs. But they do not know any secret keys shared between the RSUs and the ITS system. *Second*, malicious users cannot control any infrastructure units or replicate them in exactly the same way. For example, malicious users cannot replicate a certain legitimate RSU by placing the same hardware on the same roadside pole. This prevents malicious users from obtaining similar profiling data as the authority does.

6.1.3 Related work

Existing location proof solutions. Location proofs have been suggested as a way for users to prove their past locations in location based services [110–112]. A typical proof construction requires a user to perform several rounds of interaction with the proof issuer to derive a location proof, which is later used by the proof verifier to verify the user's location. Later work by [112] improves on this process by preventing the proof issuer from learning the user's location. We will discuss the limitations of applying the existing location proof solutions in vehicular environments in details later.

Location privacy in vehicular networks. To prevent users from submitting fake information in vehicular networks, the existing solutions typically use anonymous authentication. Work by Xi et al [107] proposes a symmetric random key-set

scheme, where each vehicles possesses a set of symmetric keys randomly chosen from a key pool, to authenticate vehicles into vehicular networks. Scheme proposed by Calandriello et al. [108] addresses the problem of anonymous message authentication using asymmetric keys and group signature in vehicular networks. ECPP [109] achieves anonymous message authentication under the help of its on-the-fly short-time anonymous keys between vehicles and RSUs.

The reasoning behind the existing approaches is that if car owners are aware that bogus data can be traced back to them, they will not intentionally upload incorrect data. However, car owners can simply opt out of providing any data if their privacy is threatened. Our work provides a technique to enable a user to prove his historical locations are in accordance with the data he reports. This allows ITS operators to prevent the attacks where users report information about places they did not visit. Our approach achieves strong user privacy protection, since we do not place any information regarding the user's identity nor link any cryptographic keys with the user, and thus there is no way users reporting data to the ITS systems can be traced. We argue that our approach is more suitable in situations where user privacy outweighs other concerns.

Concerns about users' location privacy have become major considerations when deploying location-related services in ITS systems [106]. Existing solutions use group navigation and dynamic pseudonyms [115], mix-zones and vehicular mix-networks [116] or group communication [117] to defend users' location privacy. VProof also well protects users' location privacy since we do not place any information regarding the user's identity nor link any cryptographic keys with the user.

Changing packet transmission power and measuring packet received signal strength. Changing wireless packet transmission power and measuring the received signal strength (RSS) have recently been used in localizations [118, 119] and rogue vehicular AP detection [120, 121]. All these existing works

rely on accurate RSS measurements to determine the distance between two communicating wireless nodes. In our work, we do not have dependence on accurate RSS measurements for the following two reasons. *First*, changing packet transmission power and measuring RSS in our solution are *not* to determine the distance between two wireless nodes. Instead, we use them to hide the inherent RSS patterns from users. Thus, our work does not need as accurate RSS readings as the existing works do. *Second*, our RSS pattern similarity comparison algorithm is specifically designed to cope with inaccurate RSS measurements and packet losses.

6.2 Motivation

6.2.1 Limitations of the current location proof solutions

Our work is motivated by the following three major limitations of the existing location proof solutions [110–112] to be applied in vehicular environments.

First, the existing solutions can only be used to prove at certain time a user was within the communication range of the proof issuer (an RSU in our case) but not at a finer granularity. This critical drawback allows a malicious user to sit tight at a certain location within the RF range of an RSU, and legitimately report bogus data about other locations within the same range. The resulting damages are even greater in systems adopting long range wireless communication techniques. In our solution, if a user claims he was at a certain place when he collected some data, we require him to show that he has seen the correct RSS pattern of the packets sent by a nearby RSU that he must *drive by the claimed place* to obtain.

Second, the construction of a location proof in the existing solutions requires several rounds of interactions between the proof issuer and the user, which makes them impractical in vehicular environments as the contact durations between vehicles and RSUs may be very short. By contrast, with VProof, no interaction be-

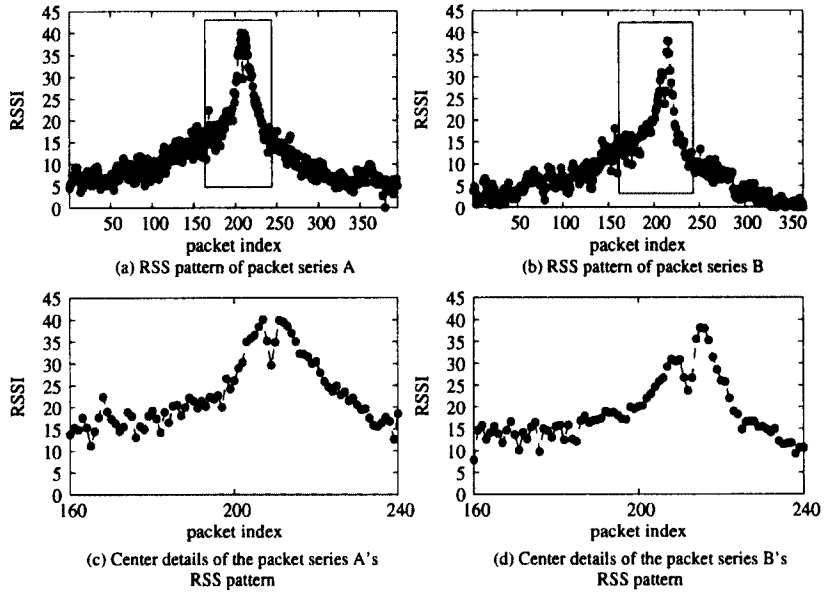


Figure 51: RSS patterns of two series of packets collected at different times. (a) and (b) show the overall patterns of the two packet series. (c) and (d) show the center details of the pattern in (a) and (b) respectively.

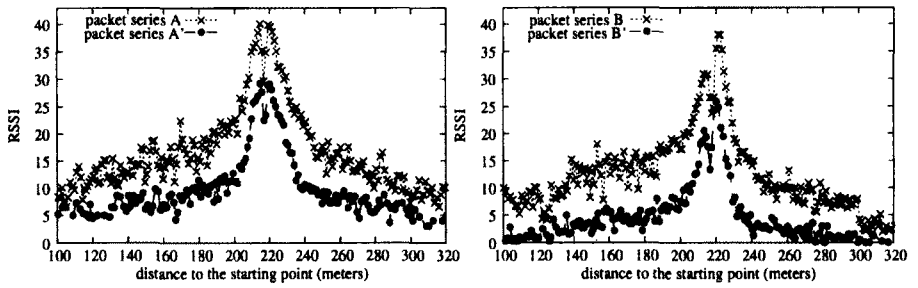
tween the proof issuer and the user is required.

Third, the existing solutions rely on PKI systems that we are trying to avoid, since the deployment of a large scale PKI scheme for ITS is unlikely to be realized in the near future.

Through real-world measurements, we make the following two key observations that led us to design our location proof solution by utilizing RSU packets RSS patterns.

6.2.2 The observation on RSS patterns of RSU packets with a fixed transmission power

Through real-world experiments, we observe that the RSS of a series of RSU packets received by a vehicle when it passes an RSU, which is continuously broadcasting packets with a fixed power, exhibit similar patterns over time. In the experiments, we deployed an wireless node, which broadcast packets at a rate of 100 packets/s with the full transmission power, at the roadside of a down-



(a) RSS patterns of packet series A and A' (b) RSS patterns of packet series B and B'

Figure 52: RSS patterns of RSU two packet series under different tx powers. Packet series A and A′ were collected in the same experiment. They were transmitted under different tx powers. Packet series B and B′ have the same relationship as that of A and A′, but they were obtained at a different time in another day.

town environment. We drove a car past the roadside wireless node and collected its packets at different times. Figure 51 shows the RSS patterns of two series of packets collected in the experiment, where series A were collected in the morning of one day when there were less cars on the road, and series B were collected during peak hours in the afternoon of another day. We can see that, although there are slight differences in RSS amplitudes and pattern shapes, which are due to factors like different temperatures and different moving obstacles (i.e., cars and trucks) on the road, the two series of packets do exhibit similar RSS patterns.

6.2.3 The observation on relatively constant RSS difference of packets with two different transmission powers

We also observe that the RSS difference between two packets that are transmitted by the same RSU using two different powers and are received at the same location is *roughly* a constant across the RSU's RF range over time. Figure 52 shows the RSS of two groups of RSU packet series. Packet series A and B, which were transmitted using the full power, are the same as in Figure 51. Packet series A′ and B′, which were transmitted using half of the full power, were obtained at the same times as A and B respectively. We align the two RSU packet series obtained in the same experiment based on the distance between each packet's

reception location and a fixed starting point. We can observe that the RSS difference under two different transmission powers at the same location is roughly the same (around 8 dbm). We will quantify how stable this RSS difference is later in Section 6.6.

6.3 Solution

6.3.1 An overview

With the above two observations, we design a location proof scheme using RSS of RSU packets, which are publicly observable. Generally speaking, we let RSUs continuously broadcast packets that are specifically for the location proof functionalities (named as "VPackets"). Each VPacket is broadcast using a *randomly* chosen transmission power. Since the transmission power is randomly selected, the RSS of the VPackets received by vehicles exhibit no pattern. Each VPacket incorporates some *encrypted* information including the transmission power of the packet. Vehicles collect the VPackets, construct location proofs based on information in the VPackets and their own GPS readings, and submit the location proofs to the ITS system for verification. Using the information in the location proofs, specifically the transmission power of each VPacket, the ITS operators can restore the inherent VPacket RSS patterns, which are the RSS patterns if the VPackets were transmitted using the full power. The location proofs are deemed as valid only if they can be used to correctly restore the inherent RSS patterns of RSUs. Since the transmission power of each VPacket is *only* known by the ITS operators, we enforce the unforgeability of the location proofs VProof constructs.

The general operation flow of VProof is shown in Figure 53. The *pre-application operations* (i.e., the operations performed before the data collection applications are deployed) include:

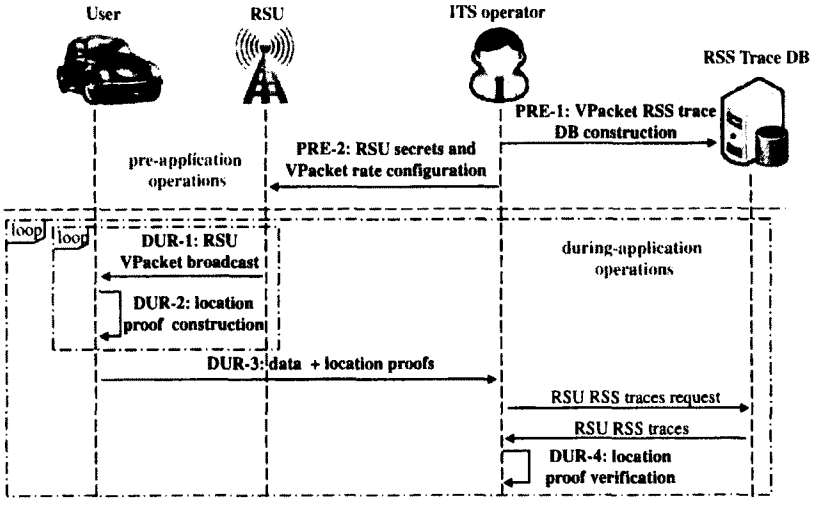


Figure 53: *Operation flow of VProof.*

- Step PRE-1: the ITS operator constructs a database that stores different RSUs' VPacket RSS traces.

- Step PRE-2: for each RSU, the operator assigns it a unique secret and configures it with a pre-calculated VPacket broadcast rate. The RSU secret is used by both the RSU and the system operator to generate/verify location proofs. The VPacket broadcast rate theoretically ensures a vehicle can receive a desired number of VPackets during each coherence time period.

The *during-application operations* (i.e., the operations performed with the data collection applications) include:

- Step DUR-1 (proof issuing): RSUs broadcast each VPacket at the configured rate using a *randomly* selected transmission power.

- Step DUR-2 (proof construction): upon receiving the VPackets, vehicles construct location proofs by extracting relevant contents from the VPackets.

- Step DUR-3 (proof submission): when vehicles upload newly collected data to the backend server, they also upload all the location proofs constructed since the last data submission.

Table 14: Major notations

Notation	Meaning
U_i, U_j	RSU IDs
N	number of traces assoc. with each trajectory in the DB
L	number of VPacket transmission powers
P_j	non-full transmission powers ($j = 1, \cdots, L-1$)
P_f	the full transmission power
$RSS_{U_i, P_f \to P_j}$	average RSS difference between VPackets transmitted under a non-full power P_j and the full power P_f
s_{U_i}	secret for U_i

- Step DUR-4 (proof verification): the ITS operator verifies the location proofs according to the following sub-steps. First, the operator verifies if the location proofs are constructed using authentic VPackets. Then he constructs a user RSS series based on the location proofs, and pre-process the user RSS series to smooth out the unpredictable vehicle moving patterns when vehicles received the VPackets, such as stops due to red lights and slow driving due to traffic jams. Before feeding the user RSS series into the RSS similarity comparison algorithm, the operator restores the inherent RSS pattern (i.e., the RSS pattern if all the VPackets were transmitted using the full power). Finally, the operator determines if the location proofs are valid based on the pattern similarities between the restored user RSS series and the DB RSS traces.

We describe the details of each of the above steps in the following section.

6.3.2 PRE-1: VPacket RSS trace database construction

The RSS trace DB contains VPacket RSS traces of each RSU. In our design, there are N RSS traces associated with each possible vehicle trajectory around each RSU U_i. Each RSS trace contains an RSS series of VPackets collected by driving a car past U_i on the trajectory. Figure 54 shows an example of the

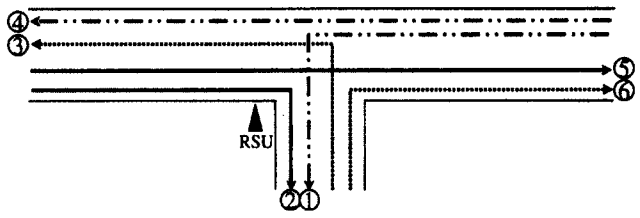


Figure 54: The six possible trajectories of a "T" shape crossing.

possible trajectories around a "T" shape crossing. During the profiling process, U_i broadcasts VPackets using the *full* transmission power. In the DB, assuming there are L transmission powers, U_i is also associated with average RSS differences between each non-full powers P_j $(j = 1, \cdots, L-1)$ and the full power P_f: $RSS_{U_i, P_f \to P_j}$. We will describe the experiment that obtains this average RSS difference in Section 6.6. The profiling process only needs to be done once.

6.3.3 PRE-2: RSU secrets and VPacket rate configuration

The system operator generates an RSU-specific secret s_{U_i} for each RSU U_i, and configures it to U_i. The secret s_{U_i} is for encrypting/decrypting the VPacket's transmission power, and also for generating/verifying the VPacket authentication message (described later). For a vehicular wireless node moving in an outdoor environment, the RSS of its received packets vary a lot. To deal with RSS instability, we want users to receive n VPackets within a period of coherence time, which is the time duration over which the RSS is considered to be not varying. Then we can take the average of these n VPackets' RSS as a data point in the RSS series. So the VPacket rate for an RSU U_i is calculated as $\frac{n}{T_{coherence}}$, where $T_{coherence}$ is the average coherence time of the road section around U_i.

6.3.4 DUR-1: VPackets broadcast (by RSU)

VPacket transmission power selection. An RSU U_i uses Algorithm 9 to select the transmission power for each VPacket when it is generated. If the timer

Algorithm 9: VPacket transmission power selection

Data: $Timer_{pwr}$ bounds T_1 and T_2 ($T_{coherence} < T_1 < T_2$).
Result: VPacket transmission power.

```
1 if Timer_pwr has expired then
2 |   p = randomly chosen from P_f, P_1, ..., P_{L-1};
3 |   d_timer = random(T_1, T_2);
4 |   Reset Timer_pwr with duration d_timer;
5 else
6 |   p = power returned when last time call this function;
7 return p;
```

$Timer_{pwr}$ is expired at the time when the algorithm is executed, the algorithm selects the transmission power randomly from the L power levels (line 2), with P_f being the full power and $P_1, \cdots, P_{L-1}$ being the other $L-1$ none-full power levels. Otherwise, the algorithm returns the power level given by the last random selection (line 6). To ensure VPackets are broadcast using the same power level during at least a period of coherence time ($T_{coherence}$), the duration of $Timer_{pwr}$ is set randomly between T_1 and T_2 (line 3), both of which are greater $T_{coherence}$.

After the transmission power p is determined, U_i encrypts p using a symmetric-key algorithm SE, with the combination of s_{U_i} and time t (i.e., the time when the VPacket is generated) as the cryptographic key:

$$C_p \leftarrow SE_{s_{U_i},t}(p). \tag{6.1}$$

VPacket authentication message generation. In order to verify if the location proofs are constructed based on authentic VPackets, an RSU U_i generates a *VPacket authentication message* (*VAM*) for each VPacket as

$$VAM \leftarrow H(U_i, s_{U_i}, t, C_p), \tag{6.2}$$

where H is a cryptographic hash function (e.g., MD5 and SHA-1) that hases U_i (the ID of the RSU), s_{U_i} (the secret of the RSU), t (the time when the VPacket is

generated) and C_p (the ciphertext of the transmission power p) into a single piece of message.

Finally, the RSU U_i puts the *VAM*, C_p and t into the VPacket, and broadcasts the VPacket using power p.

6.3.5 DUR-2,3: Location proof construction and submission (by vehicles)

Upon receiving a VPacket, a vehicle constructs a location proof (LP) as

$$LP =< U_i,\ \mathit{VAM},\ t,\ \mathit{RSS},\ C_p,\ \mathit{LOC} >, \tag{6.3}$$

where (1) U_i is the ID of the VPacket's originating RSU, (2) *VAM* is the VPacket authentication message, (3) t is the time when the VPacket is generated, (4) *RSS* is the received signal strength of the VPacket, (5) C_p is the ciphertext of the VPacket transmission power p, and (6) *LOC* is the vehicle's GPS location when the packet is received. The first five items are extracted from the VPacket, and the last one is obtained from the vehicle's onboard GPS device.

When the user uploads newly collected data to the ITS system, he also uploads all the location proofs constructed since the last data submission to the ITS system.

6.3.6 DUR-4: Location proof verification (by ITS operators)

The ITS operator divides the location proofs received from the *same* upload connection into batches such that each batch of location proofs share the same RSU ID. Then the operator verifies the location proofs batch by batch. A valid batch of location proofs with RSU ID U_i indicate the proof submitter has actually driven past U_i at the time indicated in the location proofs.

VPacket authentication message verification. Given a batch of m loca-

tion proofs, the ITS operator verifies the *VAM* contained in each location proof as follows. He computes the message content by using formula (6.2) with the parameters U_i, t, C_p and s_{U_i}, where U_i, t and C_p are extracted from the location proof, and s_{U_i} is kept by the operator. If the computed content is different from the *VAM* contained in the location proof, the *VAM* is deemed as invalid. An invalid *VAM* indicates that a least one parameter of U_i, t and C_p provided in the location proof has been tampered with. If there exists one location proof containing invalid *VAM*, the whole batch of location proofs are invalid. Note that a batch of location proofs with valid *VAM*s does not necessarily means the batch of location proofs are valid, because a malicious user can *statically* collect VPackets, obtain valid *VAM*s and present them in the location proofs.

User RSS series construction. Given a batch of m location proofs with valid *VAM*s, the operator constructs a user RSS series S_{user} according to Algorithm 10. In the algorithm, S_{raw} is the sequence of RSS in the m location proofs, T_{raw} is the corresponding sequence of VPacket generation times, and P_{raw} is the corresponding sequence of deciphered transmission power levels. The algorithm outputs the user RSS series S_{user}, in which each data point is computed as the *average* RSS (line 7) of the VPackets that were transmitted under the same power and within a period of coherence time (line 6). The algorithm also outputs $P_{S_{user}}$, a sequence of deciphered transmission power levels. Each data point of $P_{S_{user}}$ corresponds to an RSS data point in S_{user} (line 8).

User RSS series preprocessing. The preprocessing of the user RSS series S_{user} has two goals. The first is to make S_{user} location-even. In real road situations, users may stop on the road for a while (due to red lights), or drive with a speed that is far less than the speed limit (due to congested traffic). In these cases, S_{user} will contain much more data points measured around some locations than from other locations. Our algorithm tunes S_{user} by removing the redundant points based on the parameters of *LOC* and t presented in the corresponding location

Algorithm 10: User RSS series S_{user} construction

Data: Raw RSS sequence: $S_{raw} = u_1 \cdots u_m$;
VPacket generation time sequence: $T_{raw} = t_1 \cdots t_m$;
Deciphered power sequence: $P_{raw} = q_1 \cdots q_m$.
Result: User RSS series: $S_{user} = v_1 \cdots v_n$;
Transmission Power sequence: $P_{S_{user}} = p_1 \cdots p_n$.

```
1  S_user ← φ, P_{S_user} ← φ;
2  for i ← to m do
3  |  if S_user == φ then
4  |  |  time ← t_i; power ← q_i; pool ← {u_i};
5  |  else
6  |  |  if (t_i > time + T_coherence) or (q_i ≠ power) then
7  |  |  |  Add average(pool) at the end of S_user;
8  |  |  |  Add power at the end of P_{S_user};
9  |  |  |  time ← t_i; power ← q_i; pool ← {u_i};
10 |  |  else
11 |  |  |  Add u_i to pool;
12 return S_user, P_{S_user};
```

proofs. The second goal of the preprocessing is to identify the user's trajectory based on which the operator can select the corresponding RSS traces from the RSS DB. The trajectory identification is based on the GPS locations contained in the proofs.

User RSS series pattern restoration. The user RSS series $S_{user} = v_1 \cdots v_n$ has no pattern since the VPackets were broadcast using random transmission powers. Therefore, before comparing the patterns of user-submitted RSS series and the profiled RSS series stored in the trace DB, the ITS operator needs to restore S_{user}'s inherent RSS pattern (i.e., the RSS pattern if all the VPackets were transmitted using the full power). This is accomplished by adding $RSS_{U_i,P_f \to P_j}$ (recall that this information is associated with U_i in the RSS DB) to each data point $v_k \in S_{user}$, if $p_k \in P_{S_{user}}$ equals to P_j, where $1 \leq j \leq L-1$ (L is the number of transmission power levels) and $1 \leq k \leq n$.

User RSS series validation. To validate the user RSS series S_{user}, the ITS operator fetches the N RSS traces associated with the user trajectory from the RSS trace DB, and derives N DB RSS series $S_{DB,i}$ ($i \in [1, N]$) in the same way as

constructing S_{user}. The ITS operators compares the similarity between the user RSS series S_{user} and each of the N profiled RSS traces respectively. The user RSS series is deemed as valid if there are enough amount of matches.

As we mentioned earlier, it is difficult to decide if two RSS series have similar patterns for the following three reasons. *First*, different hardwares may have different readings on the same received packet, because they may have different noise floors. To address this issue, our RSS series comparison algorithm is designed to compare patterns of the *quantized* series, which are not impacted by amplitudes of the RSS readings. *Second*, RSS measurements are sensitive to many factors especially in an outdoor moving environment. *Third*, due to busy vehicular wireless environment, vehicles may not received all the VPackets based on which the location proofs are constructed. To address the second and the third challenges, we designed a *dynamic time warping* (DTW) [122, 123] based algorithm to compare two RSS series.

Our RSS series similarity comparison algorithm compares the user RSS series S_{user} with each of the N DB RSS series $S_{DB,i}$ according to the following steps.

The *first* step is to quantize both the RSS series S_{user} and $S_{DB,i}$ using a K-number alphabet. The goal of the RSS series quantization is to, as pointed out previously, remove the factors that can cause different amplitudes on RSS readings (e.g., hardware differences). The quantized value of each data point in a RSS series reflects the position of data point's value within the value range of the RSS series. The quantization algorithm is given in algorithm 11. We use a simple example to illustrate how the quantization process works. Suppose there is an RSS series whose RSS values are in the range of $[1, 30]$, and we want to quantize them using a 3-number alphabet $\{0, 1, 2\}$ (i.e., $K = 3$). Then our algorithm converts the data points in the RSS series with values from in the ranges of $[1, 10]$, $[11, 20]$ and $[21, 30]$ to numbers 0, 1 and 2 respectively.

The *second* step is to obtain the warped versions of the quantized RSS se-

Algorithm 11: RSS series quantization using a K-letter alphabet

Data: An RSS series $S = a_1 \cdots a_n$
Result: Quantized version of S: $Q = b_1 \cdots b_n$

```
Q ← φ;
max ← maximum(a_1, ··· , a_n);
min ← minimum(a_1, ··· , a_n);
for i ← 1 to n do
    for j ← 1 to K do
        if a_i < min + ((min−max)×j)/K then
            Add j to the end of Q;
            Break the j-loop;
return Q;
```

ries. The goal of this step is to cope with inaccurate RSS measurements and potential missing data points due to losses of VPackets. In this step, both quantized S_{user} and quantized $S_{DB,i}$ are converted to their corresponding warped versions using a dynamic time warping (DTW) based algorithm. Here we use a detailed example to show how our DTW algorithm works and its benefits. Suppose M ="1222221100" and N ="1022110000" are two RSS sequences quantized with a 3-number alphabet {0,1,2}. We can see that the two RSS sequences have the similar pattern (they both change follow the $1 \rightarrow 2 \rightarrow 1 \rightarrow 0$ pattern). However, the quantized values are not exactly aligned. For example, the second quantized value in the sequence N is 0 instead of 1 or 2, which can happen due to inaccurate RSS measurements. Meanwhile, we can see that the number of some quantized values in a sequence is less than that in another sequence (for instance, M has less 0 and N has less 2). This can happen when there are VPacket losses. We use the following formula to calculate the distance between M and N as $distance(M, N) = \sum_{i=1}^{l} |m_i - n_i|$, where m_i and n_i are the values of the i-th bit in M and N respectively, and l is the length of both sequences. The distance between the original M and N is 6 (Figure 55 (a)). Our algorithm calculates a warping path between M and N by using dynamic time warping, which is basically a form of dynamic programming, and converts M and N to their warped

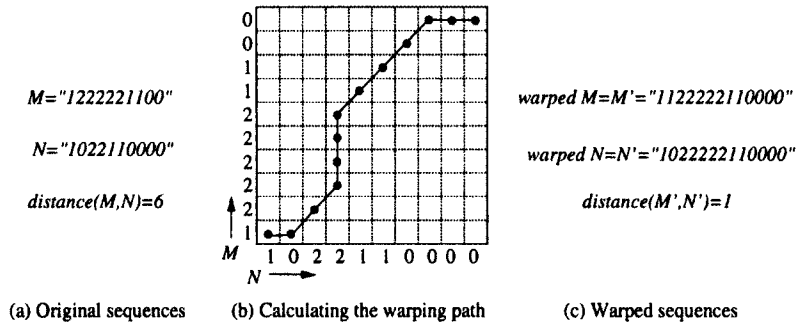


Figure 55: Converting the original sequences to warped sequences.

versions M' and N' based on the warping path (Figure 55 (b)). The distance of the two warped series is 1 (Figure 55 (c)). Now we can see that the DTW algorithm can well identify the similar pattern of two misaligned sequences. Please note that although in the example the two sequences have the same number of bits, it is not necessary that the two original sequences should have the same length. However, the two warped series will be equal-length, because both of them are constructed based on the same warping path.

In the *final* step, the algorithm calculates the similarity score between the two RSS series, based on which a conclusion is drawn. Suppose the length of the two equal-length quantized warped RSS series S'_{user} and $S'_{DB,i}$ is l, the similarity score between S'_{user} and $S'_{DB,i}$ is defined as

$$sim\_score = 1 - \frac{hamming\_dist(S'_{user}, S'_{DB,i})}{l}, \tag{6.4}$$

where the function $hamming\_dist(\cdot)$ calculates the hamming distance, which is the number of different bits, between the two input sequences. If the score is higher than a threshold s_{thresh}, the user RSS series S_{user} is considered to be similar to the i-th DB RSS series $S_{DB,i}$. We will empirically identify the suitable value of s_{thresh} later in Section 6.6. Within the comparisons between S_{user} and the N DB RSS series, if there are no less than $n_{thresh} \cdot N$ matches, the batch of location proofs are identified as valid.

6.4 Threat prevention analysis

VProof preserves privacy for users participating the data collection applications, since it does not require any information that can be related to a user's ID. VProof can efficiently prevent the threat that a malicious user reports data about location $\mathcal{L}$'s conditions at a certain time $\mathcal{T}$ without presenting at $\mathcal{L}$ at time $\mathcal{T}$. Let's consider the following four cases of the threat.

Case I. The malicious user did not drive past the location $\mathcal{L}$ at time $\mathcal{T}$. In this case, the malicious user cannot obtain the valid VPacket authentication message (*VAM*), which will make their location proofs rejected in the first step in the proof verification process.

Case II. The malicious user drove past the location $\mathcal{L}$ at time $\mathcal{T}'$, and saved the received VPackets, based on which he constructed location proofs for another time $\mathcal{T}$. In this case, the location proofs will be rejected, because the *VAM*s are not valid (recall that the construction of *VAM* depends on several parameters including time).

Case III. The malicious user drove past the location $\mathcal{L}'$ at time $\mathcal{T}$, and saved the received VPackets, based on which he constructed location proofs for another location $\mathcal{L}$, where $\mathcal{L}$ and $\mathcal{L}'$ are covered by signals of two different RSUs respectively. In this case, the location proofs will be rejected, also because the *VAM*s are not valid (they are constructed using different RSU IDs and RSU secrets).

Case IV. The malicious user *statically* collected VPackets at location $\mathcal{L}'$, where $\mathcal{L}$ and $\mathcal{L}'$ are covered by signals of the same RSU, around time $\mathcal{T}$. This is the case the existing location proof solutions cannot prevent. By contrast, VProof can easily reject those location proofs, because the restored pattern of the RSS series constructed from the statically collected location proofs cannot match that of the RSS traces stored in the RSS DB. Meanwhile, the malicious user cannot tune the parameter *RSS* contained in the location proofs so that the restored RSS pattern

matches RSS DB's record, since the malicious user has no knowledge about the VPacket transmission power, which is randomly selected and only known by the operator. We will empirically evaluate the case that a malicious user submits location proofs constructed with valid *VAM*s but guessed *RSS* in the next section.

6.5 Discussion

Privacy vs. accountability. VProof provides a new option of location proofs with strong user privacy protection. We realize that there is a tradeoff between privacy and accountability. For instance, in VProof, ITS operators cannot trace malicious users if there exist some, since there is no way users submitting data can be traced. Although this approach sacrifices the ability to trace down malicious users reporting fake data, it greatly protects users' privacy, and thus encourages participation of data collection. We argue that our solution is more suitable for the situations where privacy concerns outweigh any other concerns, as car owners can simply opt out of providing any data if their privacy is threatened.

Reporting fake information with valid location proofs. By verifying if a user's historical locations match the data he submits, we prevent the threat that a malicious user reports fake information about numerous places he did not actually visit. However, our solution does not deal with the threat that malicious users report fake information with valid location proofs (i.e., the malicious users drive their cars around a certain area, collect VPackets, construct valid location proofs and report fake information about that area). We consider this problem to has less impacts than the threat we are considering for the following three reasons. *First*, this problem is equivalent to the case where honest users unintentionally submit data sensed by defective data collection devices. *Second*, compared to the threat of interest where the malicious users can influence as many places as he wants, reporting fake information with valid location proofs can only impact a

small amount of places where the malicious users have actually visited. *Third*, the malicious users have much greater difficulties to report fake information with valid location proofs than to report fake information about places without actually visiting there, since instead of sitting somewhere statically, they need to constantly drive around the area of interest to launch the attack.

Attacks by offline profiling inherent RSS pattens. Another possible way to compromise our scheme is that malicious users first obtain RSS traces similar to those stored in the RSS trace DB by offline profiling, then they construct valid location proofs by applying RSU transmission power change histories to the profiled RSS traces. To launch the suggested attack, malicious users first need to obtain similar RSS patterns as the system operators do, which is hard to achieve, because we assume malicious users cannot replicate infrastructure units (such as RSUs). Moreover, to launch the suggested attack, a malicious user also has to stay within the radio range of an RSU, and learns its transmission power change history. Therefore, the damage caused by the attack only covers the radio range of that particular RSU. Consequently, our scheme can still prevent malicious users from reporting fake information about a large number of places that they never visited.

Wormhole attacks. An wormhole attack is launched by two colluding attackers. In this attack, an attacker drives in the communication range of an RSU, collects authentic VPackets from the RSU and delivers the VPackets to another remote attacker. Then the remote attacker reports fake information with valid location proofs constructed from the authentic VPackets. In this case, the wormhole attack is equivalent to our previous discussion point (i.e., reporting fake information with valid location proofs), as we can treat the two colluded attackers as one attacker.

6.6 Evaluation

6.6.1 Implementation and experimental setup

The prototype system. Our prototype VProof system consists of an wireless access point (Wiligear WBD-500 integrated radio board) mounted at roadside that serves as an RSU, a vehicular node equipped with a wireless receiver (Lenovo T61 + wireless card + external omni-directional antenna) that acts as ITS system users and a backend server (Dell T3500) that performs the operations done by the operator. The wireless AP runs a program that controls VPacket transmission power and broadcast rate according to the VProof scheme. The vehicular wireless node runs a program that constructs location proofs when VPackets arrive. To study whether hardware differences at user side have impacts on our scheme, we have used two different wireless cards on the vehicular wireless node, Ubiquiti SWX-SRC and Wistron CB9-GP, both of which have an external antenna socket. The backend server processes the location proofs constructed by the vehicular node offline. Note that in real ITS data collection applications, the sensed data and meta-data are not required to be uploaded to the backend server in real time [124–126]. Users can choose to upload the data anytime they feel appropriate, for example, when the uploads will not contend with other important tasks. Therefore, our choice of letting the backend server process the location proofs offline conforms the reality.

Per-packet transmission power control. The program we ran on the wireless AP is able to change the transmission power for each packet. This per-packet power control is achieved by specifying the desired transmission power in the packet's radiotap structure. With the per-packet transmission power control, it is possible to change VPacket's transmission power randomly while not affecting the normal tasks done by the RSU (e.g., beacon broadcasting).

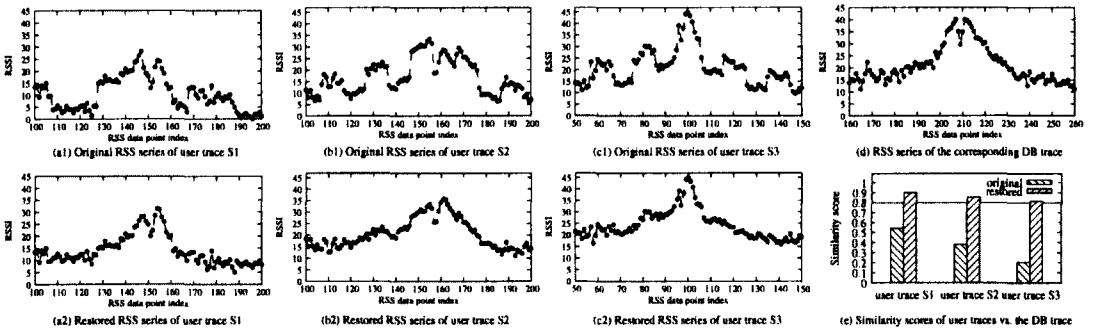


Figure 56: RSS patterns comparisons of RSS series collected at the road section A.

Experimental setup. We have conducted extensive experiments on real road situations to evaluate our solution. The experiments were performed at three different locations in downtown environment with busy road traffic. We built the RSS DB by profiling the RSU RSS pattern (i.e., we drove car past the RSU, which was broadcasting VPackets using the full transmission power, and collected the VPackets) at the three locations in one day. We then performed experiments to collect user RSS traces (i.e., we drove car past the RSU, which was broadcasting VPackets using randomly selected transmission powers, and collected the VPackets) at each location in three other days. Three RSU transmission power levels were used: full power, $\frac{3}{4}$ power and half power. Note that all the three tested cases involved relatively straight routes. We expect our scheme works similarly for cases containing turnings and other irregular routes. This is because VPacket RSS pattern of a particular route is relatively stable as long as the surrounding structures remain unchanged. Furthermore, the driving speed in our experiment was around 20 mph to 30 mph. We expect our solution also works in high speed scenario. This is because VPacket rate by RSUs is adapted dynamically based on the average road speed: recall that VPacket rate is calculated based on the coherence time of the environment, which is closely related to speed limit of the road.

6.6.2 Experimental results

Coherence time measurement. We measured the coherence time of the wireless channel at each of the three experiment locations as Camp and Knightly [127] did. Specifically, we let the RSU broadcast small packets (100 bytes/packet) at a very high packet rate (500 packets/s). Then we measured the RSS differences between different size of VPacket windows based on which we determined the coherence time of the channel. We found that the coherence times at our experiment locations were around 100 ms. Thus in our experiments, we set the VPacket rate to 100 frames/s, as we wanted users to receive 10 VPackets per period of coherence time. Meanwhile, according to Algorithm 10, when constructing the RSS series, we computed an RSS data point as the average RSS of VPackets that are transmitted under the same power and within 100ms.

Quantifying the stableness of RSS difference of VPackets with two different powers. To restore the inherent RSS pattern from the RSS of VPackets broadcast using random transmission powers, our solution relies on the fact that the difference of RSS at the same location between two VPackets that are transmitted by the same RSU using two different powers is roughly a constant across the entire communication range of the RSU. We conducted an experiment to quantify how stable this RSS difference is. In the experiment, we let the roadside wireless AP change between two transmission powers periodically while broadcasting the VPackets. In the first half of a coherence time period, the VPackets were transmitted using the full power, and in the second half, the VPacket transmission power was set to half of the full power. The VPacket rate was 100 frames/s. We drove a car past the RSU and collected the VPackets. We processed the received VPackets by first averaging the RSS of consecutive VPackets with the same transmission power. Then we calculated the RSS difference between two sets of VPackets that were transmitted with two transmission

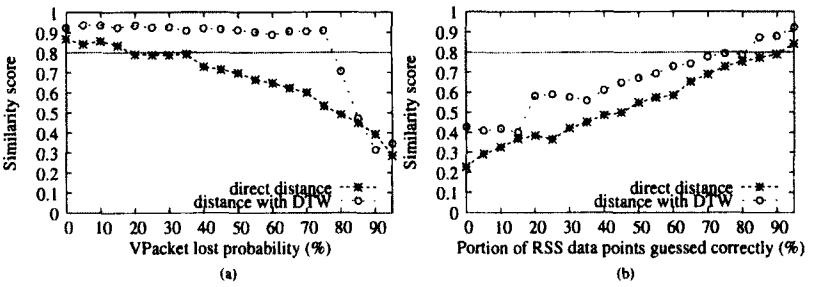
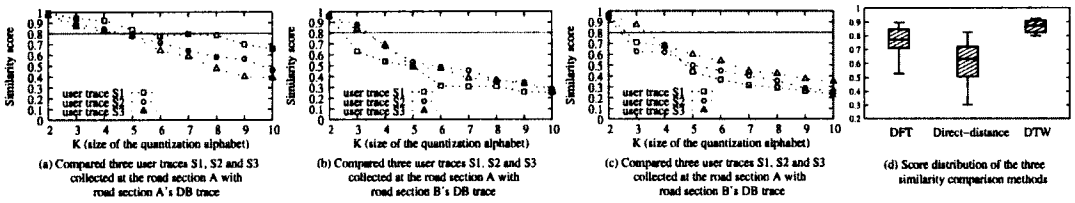


Figure 57: Dealing with VPacket losses and guessed series.

powers within a period of coherence time. Finally, we found that the average RSS difference is 8.55 dBm with a standard deviation of 2.81 dBm. As we will show later, this RSS difference is already stable enough that allows us to keep the fidelity of RSS patterns during the pattern restoration process.

RSS pattern comparisons. We have collected over 50 user traces at the three experiment locations, which are denoted as road section A, B and C respectively. When comparing these honest user traces to their corresponding DB traces, we used an alphabet of four numbers (i.e., $K = 4$) at the quantization step. Empirically, we found that the similarity scores between all the user traces and the corresponding DB traces are larger than 0.8. Thus in our scheme, we set the threshold similarity score s_{thresh} as 0.8.

We show the RSS pattern comparisons for traces collected at the road section A. Figure 56 (a1), (b1) and (c1) show the patterns of three original RSS series constructed from three user traces, $S1$, $S2$ and $S3$, that were collected at the road section A. Figure 56 (a2), (b2) and (c2) show the patterns of the corresponding restored RSS series. Figure 56 (d) shows the pattern of the RSS series constructed from the DB's trace. We have only show the center part of the patterns for better illustration. The randomly selected transmission power of VPackets allows the RSS of VPackets received within a short range of distance exhibit no fixed pattern. Therefore, we can see that the three original RSS series are all different (the similarity scores of any pair of the three original RSS series are less than 0.58). However, once we restore the original RSS series to their corresponding



(a) Compared three user traces S1, S2 and S3 collected at the road section A with road section A's DB trace

(b) Compared three user traces S1, S2 and S3 collected at the road section A with road section B's DB trace

(c) Compared three user traces S1, S2 and S3 collected at the road section A with road section B's DB trace

(d) Score distribution of the three similarity comparison methods

Figure 58: Determining quantization alphabet size K and comparing different similarity comparison methods.

full power RSS series, the three user traces do exhibit similar RSS patterns with the DB trace. Figure 56(e) shows the similarity scores when comparing the DB trace against the original RSS series and against the three restored RSS series. It shows that the similarity scores between each original RSS series and the DB's RSS series are smaller than 0.55, and the similarity scores between each restored RSS series and the DB's RSS series are all larger than 0.81.

Another point worth mentioning is that of the first two user traces, $S1$ and $S2$, and the DB trace were collected using the Ubiquiti wireless card, while the third user trace $S3$ was collected using the Wistron wireless card. Although the RSS readings from the Wistron card were higher, it did not prevent our solution from correctly accepting the honest user trace, since our similarity comparison scheme only depends on the pattern itself but not RSS reading amplitude.

Dealing with VPacket losses. In a busy vehicular wireless environment, vehicles may not receive all the VPackets, based on which the location proofs are constructed. Our scheme addresses this issue by letting RSU broadcast multiple VPackets within a coherence time period and designing a DTW-based algorithm that can effectively deal with missing data points when comparing two RSS series. However, the VPacket losses scenario was not fully manifested in our real-world experiments, since we did not have a busy wireless environment on the road. To evaluate how our solution deals with VPacket losses, we simulated the losses by randomly taking VPackets out from an honest user trace based on certain probability. Then we constructed the RSS series from the user trace with missing

VPackets, and compared it with the corresponding DB RSS series. Figure 57(a) shows the performances of the direct distance algorithm (i.e., directly compared two series without performing the DTW process) and the DTW algorithm in dealing with traces with missing VPackets. We can see that the DTW algorithm keeps the similarity score between the honest user trace with missing VPackets and the corresponding DB trace staying above the acceptance threshold as the VPacket lost probability goes up to 70%. In other words, for an honest user trace, as long as 3 of the 10 VPackets sent within the coherence time are received, our DTW algorithm can correctly mark it as valid.

Dealing with guessed RSS series. In this experiment, we considered the case that a malicious user submits location proofs with guessed RSS values in the hope that the restored RSS pattern can match the DB RSS traces. We first constructed an RSS series by randomly generating RSS values that are within the RSS range in our experiments. Then we simulated the malicious user has correctly guessed the value of an data point by replacing a random data point of this guessed series with a corresponding real data point in one of our user RSS series. We increased the proportion of the replacement to see how our scheme responds. Figure 57(b) shows the result. We can see that for a malicious user to get his RSS series accepted, he needs to correctly guess more than 80% of the data points. Since there are usually hundreds or thousands of data points in an RSS series (as in our experiments), we can draw a conclusion that the probability of a malicious user successfully makes a guessed RSS series accepted by our scheme is very low.

Evaluating RSS series quantization. This experiment evaluates how the quantization alphabet size K affects the similarity score of two RSS series. We consider both the case of matched RSS series and the case of mismatched RSS series. For the matched case, we compared $S1$, $S2$ and $S3$, which are three user RSS traces collected at the road section A, with the DB RSS trace of road

section A (Figure 58 (a)). For the mismatched case, we compared $S1$, $S2$ and $S3$ with the DB traces of the road section A and road section B respectively (Figure 58 (b) and (c)). We can see that when choosing 4 and 0.8 as the values for parameters K and s_{thresh}, our scheme can correctly identify the RSS patterns of traces collected at the same location and also correctly distinguish the RSS patterns of traces collected at different places. According to our experimental results, we can develop an empirical formula to determine the value of K: $K = 3 + \lfloor (0.9 - s_{thresh}) \times 10 \rfloor$.

Evaluating different similarity comparison methods. Our solution applies a DTW algorithm in the RSS series similarity comparison scheme. For a good similarity comparison scheme, the range of similarity scores it generates when comparing two similar series should be as narrow as possible, because a wide range of scores would make it hard to determine the similarity threshold, and it would likely lead to false positives or false negatives.

In this experiment, we compare *three* methods in terms of their ability to produce narrow score range when comparing two similar series. The *first* method dynamic time warping (DTW), where the similarity score is calculated based on the warped versions of the two quantized RSS series. The *second* method is direct distance (DD), where the similarity score is calculated directly based on the two quantized RSS series. The *third* method we compare is the Discrete Fourier Transform (DFT) method. DFT is a classical method to compare the similarities between time-series [128, 129]. With this method, DFT is first applied to the two time-series to compare. Then the two series are represented by the $2f_c$ coefficients of the first f_c frequencies of their DFT result. Here f_c is called the "cutoff frequency". Then the similarity is quantified as the distance between the two $2f_c$-dimension vectors. We have tried this method as the RSS series comparison method for our scheme. Suppose after applying DFT, two RSS series S_1 and S_2 are presented as two $2f_c$-dimension vectors $< \alpha_{f_1,cos}, \alpha_{f_1,sin}, \cdots, \alpha_{f_c,cos}, \alpha_{f_c,sin} >$

and $< \beta_{f_1,cos}, \beta_{f_1,sin}, \cdots, \beta_{f_c,cos}, \beta_{f_c,sin} >$, the similarity score between S_1 and S_2 is computed as

$$sim\_score = 1 - \frac{\sum_{i=1}^{f_c} \sqrt{(\alpha_{i,cos} - \beta_{i,cos})^2 + (\alpha_{i,sin} - \beta_{i,sin})^2}}{\sum_{i=1}^{f_c} BIGGER(m_{S_1,i}, m_{S_2,i})}, \quad (6.5)$$

where $m_{S_k,i}$ $(k = 1, 2, i \in [1, f_c])$ is the magnitude of the i-th frequency in the DFT result of series S_k, i.e., $m_{S_k,i} = \sqrt{\theta_{i,cos}^2 + \theta_{i,sin}^2}$, here $\theta_{i,cos}/\theta_{i,sin}$ are $\alpha_{i,cos}/\alpha_{i,sin}$ if $k = 1$, and $\theta_{i,cos}/\theta_{i,sin}$ are $\beta_{i,cos}/\beta_{i,sin}$ if $k = 2$; and function $BIGGER(\cdot)$ that returns the bigger value of its parameters.

We used the three methods (DTW, DD and DFT) to compare 25 of the (honest) user traces against their corresponding DB traces. The distribution of the computed similarity scores is plotted in Figure 58 (d). In the figure, the top/bottom whiskers show the maximum/minimum values of the similarity scores. The top/bottom of the boxes represent the upper/lower quartiles of the similarity scores, and the bars within the boxes represent the mean values of the scores computed by the three methods. Through this figure we can see that the DTW method outperforms the other two methods in that it produces the narrowest range of similarity scores, which makes it most suitable to serve as our RSS series comparison method.

6.7 Conclusion

VProof is a lightweight and privacy-preserving location proof solution that does not rely on PKI systems. We built a VProof prototype system and evaluated it with extensive experiments performed on actual road conditions. The evaluation results show that VProof can effectively verify if users' location claims match their historic locations without harming their location privacy.

7 Conclusion and Future Work

In this dissertation, we propose our work of providing efficient services for mobile applications in smartphones. We have conducted five projects to improve the efficiency of three major services for smartphone applications: wireless communication service, power management service, and location reporting service. In this chapter, we summarize the five projects, and introduce our experience and lessons learned from each of the them, as well as from five of them as a whole.

We first presented the details of ETCH, which improves the efficiency of wireless communication service by enhancing spectrum utilization efficiency for cognitive radio communications. High channel availability is the key advantage brought by the cognitive radio technology. Therefore, when we design algorithms or systems for cognitive radio networks, we should keep this in mind so that we can fully take advantage of the benefits of cognitive radios. In our ETCH project, the core idea is to spread the load of communication rendezvous to all the rendezvous channels so that the wide channel availability in cognitive radio networks can be fully utilized. Our evaluation shows that, by fully taking advantage of the high channel availability, our solution achieves better throughput, requires less time to rendezvous, and scales better when there was a large amount of concurrent communications.

For the power management service in smartphones, we presented HoWiES, CacheKeeper, and an accurate CPU power modeling approach for multicore smartphones, each of which improves the power management service from a different

perspective. HoWiES achieves energy savings for WiFi communications in smartphones by exploiting low-power ZigBee radio interface. Wireless communication energy efficiency is one of the most important factors that affect smartphone user experiences, because almost every smartphone application requires some sort of wireless communication. Our study shows that it is promising to exploit the heterogeneous radio capabilities to achieve wireless communication energy saving in smartphones. In HoWiES, we utilize energy-efficient ZigBee radio to perform WiFi operations that are not forming real communications, and therefore allow WiFi radio to stay in low power mode to conserve energy during those WiFi operations. The key technical challenge is to build a method to enable communication between WiFi and ZigBee. Our WiFi-ZigBee message delivery scheme is an effort that addresses this challenge. Prior to our project, most works on heterogeneous radio coexistence were to minimize interferences between incompatible radios. Our work shows that the notion of facilitating communication between heterogeneous radios can also bring great benefits. Beside energy saving, we plan to exploit this notion in other scenarios like smart home applications in the future.

CacheKeeper is a system-wide HTTP caching service for reducing energy consumption of web based applications. The source of the energy saving is the imperfect web caching we found in many smartphone applications through an extensive and systematic Android application survey. The causes of the imperfect web caching are twofold: lack of library support and ignorance from developers. To provide correct web caching for smartphone application while taking the development burden off from developers, we propose to provide web caching as a system service in smartphones. CacheKeeper is a system we designed and implemented for this purpose. Our real-world evaluation shows that our system can significantly improve the performance and energy efficiency of smartphone applications without changing them. The lesson we learned from this project is that it is worthwhile to implement a solution as a system service, if the problem

that the solution tends to solve is widespread with user space applications.

Our CPU power modeling project, to the best our knowledge, is the first effort in exploring CPU power modeling in smartphones. In this project, we have found that the existing approaches for modeling CPU power consumption are not well suited for modern multicore smartphones. The root cause is that the existing approaches do not consider the impacts of CPU idle power states. In fact, CPU idle power states, as we showed in our experiments, could significantly affect CPU power consumption. Therefore, we designed an idle-power-state-aware approach to model CPU power consumption in smartphones.

For the location reporting service, we presented VProof, which is a lightweight and privacy preserving location proof scheme for smartphone applications in the context of vehicular networks. Location proof is important to the proper functioning of the emerging location based service (LBS) applications in smartphones. In this project, we have learned that the existing location proof solutions cannot well protect users' location privacy, and cannot scale well to large amount of users. To address these limitations, we propose a scheme that utilizes received signal strength of wireless communication packets as location proofs, and uses dynamic transmission power at the APs to achieve the security and privacy guarantees.

Besides the detailed experiences that are specific to each of the projects, we have also learned several lessons from this dissertation work as a whole. In the following, we discuss two of them.

First, we learned that at the early stage of a research project, it is important to conduct a systematic real-world study to thoroughly understand the problem to tackle. On one hand, this type of study can give us deeper understanding about the sources of the problem, which in turn would help the problem-solving. For example, in the HoWiES project, our extensive measurement study helped us find the three scenarios for WiFi energy saving. In the CacheKeeper project, the large-scale Android application survey helped us understand the imperfect

web caching problem from the perspective of individual smartphone applications. In the solution design of CacheKeeper, we needed to address the same-click redundancy that we found in the survey. On the other hand, this early stage study can also lead us to the foundation of the solution. For instance, in the VProof project, our measurement study in vehicular networks allowed us to find the two important observations that eventually served as the foundation of our RSS-based *location proof scheme*.

Second, we learned that it is meaningful and rewarding to reexamine the "old" topics in new settings (e.g., with a new application/service, with a new hardware). For example, our CacheKeeper project essentially studied a decades-old research problem: web caching. It seemed that every aspect of web caching has been well studied. However, as smartphone applications are getting popular, web browser is no longer the single source of web content consumer in smartphones - applications made by different developers are also the major sources of web traffic. This new trend brought a new problem on how different applications behave in web caching, which was the target of our Android application survey. Another example is in our smartphone CPU power modeling project, the problem of modeling CPU power consumption itself has been widely studied in the x86 architecture. Many solutions have been proposed because of that. However, as we discussed previously, these solutions cannot be directly applied to the ARM architecture, which is the dominant architecture in smartphones. Therefore, it is worthwhile to reexamine the CPU power modeling problem in the context of smartphones.

7.1 Future work

We plan to continue our research of providing efficient services for smartphones such that mobile applications can run more smoothly and consumes resources more effectively. Specifically, we will focus our attention on the following direc-

tions.

First, connectivity is the key factor that enables mobile and ubiquitous computing, and also the main reason for the popularity of smartphones. Therefore, we would like to invest more efforts in improving the wireless communication service in smartphones. For example, so far our research has been focusing on improving communication performance between phone and phone or between phone and infrastructure. As wearable devices, such as smart watches and smart glasses, are getting popular, systems that integrate smartphones and wearable devices are receiving more attention. We are interested in exploring how to provide efficient communication between smartphones and wearable devices. Meanwhile, as cloud computing is getting mature, the desire of integrating mobile computing and cloud computing is becoming prominent. We plan to investigate improving communication efficiency between smartphones and the cloud under different cloud services.

Second, we plan to continue our research in improving the power management service for smartphones, especially for the increasingly popular multicore smartphones. Though power management of multicore CPUs has been studied for years for desktops and servers (with x86 architecture CPUs), the research results there could barely fit to mobile devices, where CPUs with fundamentally different architectures (e.g., ARM, MIPS) are being used. For example, as we showed in Chapter 5, hardware events based CPU power modeling approaches, which are proposed by many x86-based works, cannot be applied in ARM-based CPUs, simply because the hardware events are not provided by ARM-based CPUs. In addition, recent advancements in mobile device multicore CPUs (e.g, octa-core design, ARM big.LITTLE technology) further allow power management in mobile devices with multicore CPUs to be a fertile ground for research projects.

Third, high level of security/privacy-protection is an equally important aspect as high energy efficiency and good performance in smartphones. Thus, offering

security and privacy monitoring services in smartphones is in our future agenda. We are interested in exploiting emerging technologies in mobile computing hardware in enhancing mobile/embedded system security and protecting user privacy. Many of these technologies may have been widely utilized for long time in ordinary (i.e., x86-based) computer systems. The different operating systems and application scenarios in mobile systems like smartphones still allow these technologies to be fairly promising in mobile system research. For example, it is timely and promising to study utilizing virtualization technology in detecting smartphone malware and protecting smartphone user privacy. Although the virtualization technology (x86-based) has been used in PC/server malware detection for about a decade, in mobile system area, hardware-assisted virtualization in the ARM architecture is available only since the unveiling of ARM Cortex-A15 MPCore on the year of 2013. Since smartphone operating systems (e.g., Android, iOS) have data/control flow design that is different from PC/server OS, there are plenty research space in applying virtualization technology in improving smartphone system security and privacy.

Bibliography

[1] A. P. Felt, K. Greenwood, and D. Wagner, "The effectiveness of application permissions," in *Proceedings of the 2nd USENIX conference on Web application development*, WebApps'11, 2011.

[2] "Google Maps." `https://play.google.com/store/apps/details?id=com.google.android.apps.maps`.

[3] "Facebook." `https://play.google.com/store/apps/details?id=com.facebook.katana`.

[4] "Foursquared." `https://play.google.com/store/apps/details?id=com.joelapenna.foursquared`.

[5] "GasBuudy." `https://play.google.com/store/apps/details?id=gbis.gbandroid`.

[6] Wikipedia, "Cognitive Radio." `http://en.wikipedia.org/wiki/Cognitive_radio`.

[7] K. Bian, J. Park, and R. Chen, "A quorum-based framework for establishing control channels in dynamic spectrum access networks," in *ACM Mobicom*, June 2009.

[8] Y. Zhang, Q. Li, G. Yu, and B. Wang, "ETCH: Efficient Channel Hopping for communication rendezvous in dynamic spectrum access networks," in *INFOCOM*, 2011.

[9] Y. Zhang, G. Yu, Q. Li, H. Wang, X. Zhu, and B. Wang, "Channel-hopping-based communication rendezvous in cognitive radio networks," *Networking, IEEE/ACM Transactions on*, vol. 22, pp. 889–902, June 2014.

[10] H. Falaki et al., "Diversity in smartphone usage," in *MobiSys*, 2010.

[11] A. Shye et al., "Characterizing and modeling user activity on smartphones: summary," in *SIGMETRICS*, 2010.

[12] E. Rozner et al., "NAPman: network-assisted power management for wifi devices," in *MobiSys*, 2010.

[13] J. Manweiler and R. R. Choudhury, "Avoiding the rush hours: WiFi energy management via traffic isolation," in *MobiSys*, 2011.

[14] Y. Zhang and Q. Li, "Howies: A holistic approach to zigbee assisted wifi energy savings in mobile devices," in *INFOCOM*, 2013.

[15] Y. Zhang and Q. Li, "Exploiting zigbee in reducing wifi power consumption for mobile devices," *IEEE Transactions on Mobile Computing*, 2014.

[16] J. Erman, A. Gerber, M. T. Hajiaghayi, D. Pei, S. Sen, and O. Spatscheck, "To Cache or Not to Cache: The 3G Case," *IEEE Internet Computing*, vol. 15, no. 2, 2011.

[17] "Study: Mobile Web Traffic Up 35% in Under a Year; PC Web Usage Peaks Early Morning." http://insights.chitika.com/2012/study-mobile-web-traffic-up-35-in-under-a-year-pc-web-usage-peaks-early-morning.

[18] "Mobile Devices Now Make Up About 20 Percent of U.S. Web Traffic." http://allthingsd.com/20120525/mobile-devices-now-make-up-about-20-percent-of-u-s-web-traffic.

[19] Y. Zhang, C. Tan, and Q. Li, "CacheKeeper: a system-wide web caching service for smartphones," in *UBICOM*, 2013.

[20] A. Carroll and G. Heiser, "An analysis of power consumption in a smartphone," in *USENIX ATC*, 2010.

[21] A. Carroll and G. Heiser, "The System Hacker's Guide to the Galaxy Energy Usage in a Modern Smartphone," in *APSys*, 2013.

[22] "Antutu Battery Saver." `https://play.google.com/store/apps/details?id=com.antutu.powersaver&feature=searchreult#?t=W251bGwsMSwxLDEsImNvbS5hbnR1dHUucG93ZXJz YXZlciJd.`

[23] R. Mittalz, A. Kansaly, and R. Chandray, "Empowering Developers to Estimate App Energy Consumption," in *MobiCom*, 2012.

[24] F. Xu, Y. Liu, Q. Li, and Y. Zhang, "V-edge: Fast Self-constructive Power Modeling of Smartphones Based on Battery Voltage Dynamics," in *USENIX NSDI*, 2013.

[25] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, "Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones," in *CODES+ISSS*, 2010.

[26] C. Yoon, D. Kim, W. Jung, C. Kang, and H. Cha, "AppScope: Application Energy Metering Framework for Android Smartphone Using Kernel Activity Monitoring," in *USENIX ATC*, 2012.

[27] D. C. Snowdon, E. L. Sueur, S. M. Petters, and G. Heiser, "Koala: A Platform for OS-Level Power Management," in *EuroSys*, 2009.

[28] M. Dong and L. Zhong, "Self-constructive high-rate system energy modeling for battery-powered mobile systems," in *MobiSys*, 2011.

[29] W. Jung, C. Kang, C. Yoon, D. Kim, and H. Cha, "DevScope: A Nonintrusive and Online Power Analysis Tool for Smartphone Hardware Components," in *CODES+ISSS*, 2012.

[30] Y. Zhang, X. Wang, X. Liu, Y. Liu, L. Zhuang, and F. Zhao, "Towards Better CPU Power Management on Multicore Smartphones," in *Hotpower*, 2013.

[31] Research and Innovative Technology Administration, US DOT, $IntelliDrive^{SM}$. `http://www.its.dot.gov/press/2010/vii2intellidrive.htm.`

[32] *CVIS: Cooperative Vehicle-Infrastructure Systems*. www.cvisproject.org.

[33] J. Eriksson, L. Girod, B. Hull, R. Newto, S. Madden, and H. Balakrishnan, "The Pothole Patrol: Using a Mobile Sensor Network for Road Surface Monitoring," in *Mobisys*, 2008.

[34] B. Hull, V. Bychkovsky, Y. Zhang, and K. C. M. Goraczko, "CarTel: A Distributed Mobile Sensor Computing System," in *Sensys*, 2006.

[35] U. Lee, E. Magistretti, B. Zhou, M. Gerla, P. Bellavista, and A. Corradi, "MobEyes: Smart Mobs for Urban Monitoring with a Vehicular Sensor Network," in *IEEE Wireless Communications*, 2006.

[36] Y. Zhang, C. C. Tan, F. Xu, H. Han, and Q. Li, "Vproof: Lightweight privacy-preserving vehicle location proofs," *IEEE Transactions on Vehicular Technology*, 2014.

[37] Y. Zhang, C. C. Tan, F. Xu, H. Han, and Q. Li, "Lightweight and privacy-preserving location proofs for intelligent transportation systems," *WM CS Technical Reposrt, WM-CS-2014-05*, 2014.

[38] J. Zhao, H. Zheng, and G. H. Yang, "Distributed coordination in dynamic spectrum allocation networks," in *IEEE DySPAN*, December 2005.

[39] P. Bahl, R. Chandra, and J. Dunagan, "SSCH: slotted seeded channel hopping for capacity improvement in IEEE 802.11 ad hoc wireless networks," in *ACM Mobicom*, September 2004.

[40] A. Tzamaloukas and J. J. Garcia-Luna-Aceves, "Channel-Hopping Multiple Access," in *IEEE ICC 2000*, 2000.

[41] I. F. Akyildiz, W.-Y. Lee, M. C. Vuran, and S. Mohanty, "NeXt generation/dynamic spectrum access/cognitive Radio Wireless Networks: A Survey," *COMPUTER NETWORKS JOURNAL (ELSEVIER)*, 2006.

[42] A. Sahai, N. Hoven, and R. Tandra, "Some Fundamental Limits on Cognitive Radio," in *Allerton Conference on Communication, Control, and Computing*, 2004.

[43] D. Cabric, S. M. Mishra, and R. W. Brodersen, "Implementation issues in spectrum sensing for cognitive radios," in *Asilomar Conference on Signals, Systems, and Computers*, 2004.

[44] A. Fehske, J. Gaeddert, and J. Reed, "A new approach to signal classification using spectral correlation and neural networks," in *IEEE DySPAN*, 2005.

[45] A. Ghasemi and E. Sousa, "Collaborative spectrum sensing for opportunistic access in fading environments," in *IEEE DySPAN*, 2005.

[46] S. Shankar, "Spectrum Agile Radios: Utilization and Sensing Architectures," in *IEEE DySPAN*, 2005.

[47] B. Wild and K. Ramchandran, "Detecting Primary Receivers for Cognitive Radio Applications," in *IEEE DySPAN*, 2005.

[48] H. Zheng and L.Cao, "Device-centric Spectrum Management," in *IEEE DySPAN*, 2005.

[49] V. Brik, E. Rozner, S. Banarjee, and P. Bahl, "DSAP: a protocol for coordinated spectrum access," in *IEEE DySPAN*, 2005.

[50] L. Ma, X. Han, and C. Shen, "Dynamic open spectrum sharing MAC protocol for wireless ad hoc networks," in *IEEE DySPAN*, 2005.

[51] L. A. DaSilva and I. Guerreiro, "Sequence-Based Rendezvous for Dynamic Spectrum Access," in *IEEE DySPAN*, October 2008.

[52] Z. Lin, H. Liu, X. Chu, and Y.-W. Leung, "Jump-stay based channel-hopping algorithm with guaranteed rendezvous for cognitive radio networks," in *INFOCOM*, 2011.

[53] Maxim Integrated Products, *Maxim 2.4GHz 802.11b Zero-IF Transceivers.*

[54] Wikipedia, *Matching (graph theory).* `http://en.wikipedia.org/wiki/Matching_(graph_theory)`.

[55] T. D. LeSaulnier, C. Stocker, P. S. Wenger, and D. B. West, "Rainbow Matching in Edge-Colored Graphs," *Electr. J. Comb.*, 2010.

[56] "Hyacinth: An IEEE 802.11-based Multi-channel Wireless Mesh Network," *http://www.ecsl.cs.sunysb.edu/multichannel/.*

[57] IEEE-SA, *IEEE Std 802.11-2007.*

[58] *ath5k wireless driver.* `http://linuxwireless.org/en/users/Drivers/ath5k`.

[59] *ath9k wireless driver.* `http://linuxwireless.org/en/users/Drivers/ath9k`.

[60] *iw configuration utility.* `http://linuxwireless.org/en/users/Documentation/iw`.

[61] A. Rahmati and L. Zhong, "Context-for-wireless: context-sensitive energy-efficient wireless data transfer," in *MobiSys*, 2007.

[62] K. Chebrolu and A. Dhekne, "Esense: communication through energy sensing," in *MOBICOM*, 2009.

[63] A. J. Nicholson and B. D. Noble, "BreadCrumbs: forecasting mobile connectivity," in *MobiCom*, 2008.

[64] G. Ananthanarayanan and I. Stoica, "Blue-Fi: enhancing Wi-Fi performance using bluetooth signals," in *MobiSys*, 2009.

[65] J. Sorber et al., "Turducken: hierarchical power management for mobile devices," in *MobiSys*, 2005.

[66] R. Zhou et al., "ZiFi: wireless LAN discovery via ZigBee interference signatures," in *MobiCom*, 2010.

[67] E. Shih et al., "Wake on wireless: : an event driven energy saving strategy for battery operated devices," in *MobiCom*, 2002.

[68] Y. Agarwal et al., "Wireless wakeups revisited: energy management for VOIP over WiFi smartphones," in *MobiSys*, 2007.

[69] IETF Network Working Group, *RFC-1191*.

[70] IEEE-SA, *IEEE Std 802.15.4-2006*.

[71] J. Liu and L. Zhong, "Micro power management of active 802.11 interfaces," in *MobiSys*, 2008.

[72] S. Ren, Q. Li, H. Wang, X. Chen, and X. Zhang, "Analyzing Object Detection Quality Under Probabilistic Coverage in Sensor Networks," in *IWQoS*, 2005.

[73] Monsoon Solutions. `http://www.msoon.com/LabEquipment/PowerMonitor`.

[74] "Android's HTTP Clients." http://android-developers.blogspot.com/2011/09/androids-http-clients.html.

[75] C. Tossell, P. T. Kortum, A. Rahmati, C. Shepard, and L. Zhong, "Characterizing web use on smartphones," in *CHI*, 2012.

[76] N. Vallina-Rodriguez, J. Shah, A. Finamore, Y. Grunenberger, K. Papagiannaki, H. Haddadi, and J. Crowcroft, "Breaking for commercials: characterizing mobile advertising," in *IMC*, 2012.

[77] I. Papapanagiotou, E. M. Nahum, and V. Pappas, "Smartphones vs. laptops: comparing web browsing behavior and the implications for caching," in *SIGMETRICS*, 2012.

[78] F. Qian, K. S. Quah, J. Huang, J. Erman, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck, "Web caching on smartphones: ideal vs. reality," in *MobiSys*, 2012.

[79] K. Zhang, L. Wang, A. Pan, and B. B. Zhu, "Smart caching for web browsers," in *WWW*, 2010.

[80] Z. Wang, F. X. Lin, L. Zhong, and M. Chishtie, "Why are web browsers slow on smartphones?," in *HotMobile*, 2011.

[81] Z. Wang, F. X. Lin, L. Zhong, and M. Chishtie, "How far can client-only solutions go for mobile browser speed?," in *WWW*, 2012.

[82] E. Koukoumidis, D. Lymberopoulos, K. Strauss, J. Liu, and D. Burger, "Pocket cloudlets," in *ASPLOS*.

[83] D. Lymberopoulos, O. Riva, K. Strauss, A. Mittal, and A. Ntoulas, "PocketWeb: instant web browsing for mobile devices," in *ASPLOS*, 2012.

[84] "Charles Web Debugging Proxy." http://www.charlesproxy.com.

[85] Fielding, et al, "RFC 2616 - Hypertext Transfer Protocol - HTTP/1.1."

[86] "Flipboard." https://play.google.com/store/apps/details?id=flipboard.app&hl=en.

[87] "Pulse News." https://play.google.com/store/apps/details?id=com.alphonso.pulse&hl=en.

[88] "Yahoo!." https://play.google.com/store/apps/details?id=com.yahoo.mobile.client.android.yahoo&hl=en.

[89] S. Guha, M. Jain, and V. Padmanabhan, "Koi: A Location-Privacy Platform for Smartphone Apps," in *NSDI*, 2012.

[90] J. Shim, P. Scheuermann, and R. Vingralek, "Proxy Cache Algorithms: Design, Implementation, and Performance," *IEEE Trans. Knowl. Data Eng.*, vol. 11, no. 4, 1999.

[91] E. W. Felten and M. A. Schneider, "Timing attacks on Web privacy," in *ACM CCS*, 2000.

[92] A. Bortz and D. Boneh, "Exposing private information by timing web applications," in *WWW*, 2007.

[93] "3G and 4G Wireless Speed Showdown: Which Networks Are Fastest?." http://www.pcworld.com/article/253808/3g_and_4g_wireless_speed_showdown_which_networks_are_fastest_.html.

[94] Hewlett-Packard and Intel and Microsoft and Phoenix Technologies and Toshiba, "Advanced Configuration and Power Interface Specification, revision 5.0," 2011.

[95] S. Kim, H. Kim, J. Kim, J. Lee, and E. Seo, "Empirical Analysis of Power Management Schemes for Multi-core Smartphones," in *ICUIMC*, 2013.

[96] S. Panneerselvam and M. M. Swift, "Chameleon: Operating System Support for Dynamic Processors," in *ASPLOS*, 2012.

[97] A. Shye, B. Scholbrock, and G. Memik, "Into the Wild: Studying Real User Activity Patterns to Guide Power Optimizations for Mobile Architectures," in *MICRO*, 2009.

[98] K. Shen, Arrvindh, Shriraman, S. Dwarkadas, X. Zhang, and Z. Chen, "Power Containers: An OS Facility for Fine-Grained Power and Energy Management on Multicore Servers," in *ASPLOS*, 2013.

[99] F. Bellosa, "The Benefits of Event-driven Energy Accounting in Power-sensitive Systems," in *ACM SIGOPS European Workshop*, 2000.

[100] R. Bertran, M. Gonzalez, X. Martorell, N. Navarro, and E. Ayguade, "Decomposable and Responsive Power Models for Multicore Processors using Performance Counters," in *ICS*, 2010.

[101] C. Isci and M. Martonosi, "Phase Characterization for Power: Evaluating Control-flow-based and Event-counter-based Techniques," in *HPCA*, 2006.

[102] ARM, "ARM Architecture Reference Manual - ARMv7-A and ARMv7-R edition, ARM DDI0406C.b,"

[103] Peter Greenhalgh, "ARM White Paper: big.LITTLE Processing with ARM Cortex-A15 & Cortex-A7," 2011.

[104] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in *IEEE 4th Annual Workshop on Workload Characterization*, 2001.

[105] "The Dolphin Browser." `https://play.google.com/store/apps/details?id=mobi.mgeek.TunnyBrowser`.

[106] R. A. Popa, H. Balakrishnan, and A. J. Blumberg, "VPriv: Protecting Privacy in Location-Based Vehicular Services," in *USENIX Security*, 2009.

[107] Y. Xi, K. Sha, W. Shi, L. Schwiebert, and T. Zhang, "Enforcing Privacy Using Symmetric Random Key-Set in Vehicular Networks," in *ISADS*, 2007.

[108] G. Calandriello, P. Papadimitratos, J.-P. Hubaux, and A. Lioy, "Efficient and robust pseudonymous authentication in VANET," in *VANET*, 2007.

[109] R. Lu, X. Lin, H. Zhu, P.-H. Ho, and X. Shen, "ECPP: Efficient Conditional Privacy Preservation Protocol for Secure Vehicular Communications," in *INFOCOM*, 2008.

[110] V. Lenders, E. Koukoumidis, P. Zhang, and M. Martonosi, "Location-based trust for mobile user-generated content: applications, challenges and implementations," in *HotMobile*, 2008.

[111] S. Saroiu and A. Wolman, "Enabling new mobile applications with location proofs," in *HotMobile*, 2009.

[112] W. Luo and U. Hengartner, "Proving your location without giving up your privacy," in *HotMobile*, 2010.

[113] B. Ostermaier, F. Dotzer, and M. Strassberger, "Enhancing the Security of Local Danger Warnings in VANETs - A Simulative Analysis of Voting Schemes," in *ARES*, 2007.

[114] M. Raya, P. Papadimitratos, V. D. Gligor, and J.-P. Hubaux, "On Data-Centric Trust Establishment in Ephemeral Ad Hoc Networks," in *IEEE Infocom*, 2008.

[115] K. Sampigethaya, L. Huang, M. Li, R. Poovendran, K. Matsuura, and K. Sezaki, "CARAVAN: Providing Location Privacy for VANET," in *ESCAR*, 2005.

[116] J. Freudiger, M. Raya, M. Félegyházi, P. Papadimitratos, and J.-P. Hubaux, "Mix-Zones for Location Privacy in Vehicular Networks," in *WiN-ITS*, 2007.

[117] A. Wasef and X. Shen, "REP: Location Privacy for VANETs Using Random Encryption Periods," in *Mobile Networks and Applications*, 2010.

[118] Y. Zhang, Z. Li, and W. Trappe, "Power-Modulated Challenge-Response Schemes for Verifying Location Claims," in *GLOBECOM*, 2007.

[119] X. Zheng, H. Liu, J. Yang, Y. Chen, J.-A. Francisco, R. P. Martin, and X. Li, "Characterizing the impact of multi-frequency and multi-power on localization accuracy," in *MASS*, 2010.

[120] H. Han, F. Xu, C. C. Tan, Y. Zhang, and Q. Li, "Defending against vehicular rogue APs," in *INFOCOM*, 2011.

[121] H. Han, F. Xu, C. C. Tan, Y. Zhang, and Q. Li, "VR-Defender: Self-Defense against Vehicular Rogue APs for Drive-thru Internet," *IEEE Transactions on Vehicular Technology*, 2014.

[122] D. J. Berndt and J. Clifford, "Using Dynamic Time Warping to Find Patterns in Time Series," in *KDD Workshop*, 1994.

[123] E. J. Keogh, "Exact indexing of dynamic time warping," in *VLDB*, 2002.

[124] U.S. Department of Transportation, "IntelliDrive Data Capture and Management Program: Transforming the Federal Role," 2010.

[125] C. Manasseh and R. Sengupta, "Middleware for Cooperative Vehicle-Infrastructure Systems," *California PATH Research Report UCB-ITS-PRR-2008-2*, 2008.

[126] P. Hu, B. Boundy, T. Truett, E. Chang, and S. Gordon, "Cross-Cutting Studies and State-of-the-Practice Reviews: Archive and Use of ITS-Generated Data," 2002.

[127] J. Camp and E. W. Knightly, "Modulation rate adaptation in urban and vehicular environments: cross-layer implementation and experimental evaluation," in *Mobicom*, 2008.

[128] R. Agrawal, C. Faloutsos, and A. N. Swami, "Efficient Similarity Search In Sequence Databases," in *FODO*, 1993.

[129] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos, "Fast Subsequence Matching in Time-Series Databases," in *SIGMOD Conference*, 1994.