ACM HotMobile 2013:

The 14th Workshop on Mobile Computing Systems & Applications

Feb 26-27, 2013
Jekyll Island, Georgia, USA

General Chair
Sharad Agarwal, *Microsoft Research, US*

Program Chair
Alexander Varshavsky, *AT&T Labs, US*

Poster & Demo Chair
Nilanjan Banerjee, *University of Maryland, US*

Student Volunteer Chair
Souvik Sen, *HP Labs, US*

Publicity Chair
Andreas Bulling, *Lancaster University, UK*

Web Chair
Christos Efstratiou, *University of Cambridge, UK*

Sponsored by

ACM, ACM SIGMOBILE, Microsoft Research, Telefonica, AT&T Labs, Google

The Association for Computing Machinery
2 Penn Plaza, Suite 701
New York New York 10121-0701

ACM ISBN: 978-1-4503-1421-3/13/02

ACM HotMobile 2013

Program

Keynote Address: Thad Starner

"Wearable Computing: Through the Looking Glass"
Associate Professor, School of Interactive Computing, Georgia Institute of Technology

Session 1: Cellular Billing

Chair: Suman Banerjee

- Splitting the Bill for Mobile Data with SIMlets
  Himanshu Raj, Stefan Saroiu, Alec Wolman, Jitu Padhye (Microsoft Research)
- Towards Accurate Accounting of Cellular Data for TCP Retransmission
  Younghwan Go (KAIST), Denis Foo Kune (University of Massachusetts Amherst), Shinae Woo, KyoungSoo Park, Yongdae Kim (KAIST)

Session 2: Power Management

Chair: Romit Roy Choudhury

- How is Energy Consumed in Smartphone Display Applications?
  Xiang Chen, Yiran Chen (University of Pittsburgh), Zhan Ma, Felix C. A. Fernandes (Samsung Telecommunication America)
- A2PSM: Audio Assisted Wi-Fi Power Saving Mechanism for Smart Devices
  Mostafa Uddin, Tamer Nadeem (Old Dominion University)
- Application Modes: A Narrow Interface for End-User Power Management in Mobile Devices
  Marcelo Martins, Rodrigo Fonseca (Brown University)

Session 3: Understanding humans

Chair: Anthony LaMarca

- The Case for Psychological Computing
  Xuan Bao, Mahanth Gowda (Duke University), Ratul Mahajan (Microsoft Research), Romit Roy Choudhury (Duke University)
- Recognizing Humans without Face Recognition
  He Wang, Xuan Bao, Romit Roy Choudhury (Duke University), Srihari Nelakuditi (University of South Carolina)
- sMFCC : Exploiting Sparseness in Speech for Fast Acoustic Feature Extraction on Mobile Devices - a Feasibility Study
  Shahriar Nirjon, Robert F. Dickerson, John A. Stankovic (University of Virginia), Guobin Shen (Microsoft Research Asia), Xiaofan Jiang (Intel Labs China)

Session 4: Sensors and data

Chair: Rajesh Krishna Balan

- Lowering the barriers to large-scale mobile crowdsensing
  Yu Xiao (Carnegie Mellon University/Aalto University), Pieter Simoens (Carnegie Mellon University/Ghent University -iMinds), Padmanabhan Pillai (Intel Labs), Kiryong Ha, Mahadev Satyanarayanan (Carnegie Mellon University)
- Open Data Kit 2.0: Expanding and Refining Information Services for Developing Regions
  Waylon Brunette, Mitch Sundt, Nicola Dell, Rohit Chaudhri, Nathan Breit, Gaetano Borriello (University of Washington)
- A Framework for Context-Aware Privacy of Sensor Data on Mobile Systems
  Supriyo Chakraborty, Kasturi Rangan Raghavan, Matthew P. Johnson, Mani Srivastava (University of California, Los Angeles)

Session 5: Panel: Mobile Systems and the Developing World

Moderator: Gaetano Borriello (University of Washington)

- Panelist: *Elizabeth M. Belding (University of California, Santa Barbara)*
- Panelist: *Lakshmi Subramanian (New York University)*
- Panelist: *Bill Thies (Microsoft Research India)*

Session 6: Mobile cloud interactions

Chair: Matt Welsh

- Cloud Displays for Mobile Users in a Display Cloud
  Lars Tiede, John Markus Bjorndalen, Otto J. Anshus (University of Tromso)
- Towards Synchronization of Live Virtual Machines among Mobile Devices
  Jeffrey Bickford (AT&T Security Research Center), Ramon Caceres (AT&T Labs - Research)
- Enabling the Transition to the Mobile Web with WebSieve
  Michael Butkiewicz, Zhe Wu, Shunan Li, Pavithra Murali, Vagelis Hristidis, Harsha V. Madhyastha (UC Riverside), Vyas Sekar (Stonybrook University)

Session 7: Vehicular networking and transportation

Chair: Lin Zhong

- Scout: An Asymmetric Vehicular Network Design over TV Whitespaces
  Tan Zhang, Sayandeep Sen, Suman Banerjee (University of Wisconsin Madison)
- Social Vehicle Navigation: Integrating Shared Driving Experience into Vehicle Navigation
  Wenjie Sha, Daehan Kwak, Badri Nath, Liviu Iftode (Rutgers University)
- Quantifying the Potential of Ride-Sharing using Call Description Records
  Blerim Cici, Athina Markopoulou (University of California, Irvine), Enrique Frias-Martinez, Nikolaos Laoutaris (Telefonica Research)

Demos

- Designing mobile advertising: User Experience factors for enhancing user adoption
  Stavros Asimakopoulos (Lancaster University), Frank Spillers (Experience Dynamics), George Boretos (Global Forecasting Solutions), Zhengjie Liu (Dalian Maritime University)
- NLify: Mobile Spoken Natural Language Interfaces for Everyone
  Seungyeop Han (University of Washington), Matthai Philipose, Yun-Cheng Ju (Microsoft Research)
- SocioPhone: Face-to-face Verbal Interaction Monitoring Using Multi-Phone Sensor Fusion
  Youngki Lee, Chulhong Min, Chanyou Hwang, Jaeung Lee, Inseok Hwang, Chungkuk Yoo, Junehwa Song (KAIST)
- Focus: A Usable & Effective Approach to OLED Display Power Management
  Tan Kiat Wee, Tadashi Okoshi, Archan Misra, Rajesh Krishna Balan (Singapore Management University)
- Mobile ICN Applications for an Event with Large Crowds
  Anders Lindgren (SICS Swedish ICT), Börje Ohlman, Karl-Åke Persson, Anders Eriksson (Ericsson Research), Petteri Pöyhönen, Janne Tuononen (Noka-Siemens Networks), Dirk Kutscher (NEC Labs Europe)
- Preserving Data Privacy Through Data Partitioning in Mobile Application
  Mohammad Al-Mutawa and Shivakant Mishra (University of Colorado Boulder)

Posters

- Bugu: An Application Level Power Profiler and Analyzer for Mobile Devices
  Youhuizi Li, Hui Chen, Weisong Shi (Wayne State University)
- Extraction algorithm of relationship between smartphone applications for recommendation
  Kohei Terazono, Akira Karasudani, Satoshi Iwata, Tatsuro Matsumoto (Fujitsu Laboratories)
- RazorCam: A Prototyping Environment for Video Communication
  Michael Mefenza, Franck Yonga, Christophe Bobda (University of Arkansas, Fayetteville)
- Mobifetch: An Execution-time Prefetching Technique to Expedite Application Launch in Mobile Devices
  Junhee Ryu, Kwangjin Ko, Heonshik Shin (Seoul National University), Kyungtae Kang (Hanyang University)
- SickleREMOTE: A Two-Way Mobile Messaging System for Pediatric Sickle Cell Disease
  Chihwen Cheng (Georgia Institute of Technology), Clark Brown (Emory University), Janani Venugopalan (Georgia Institute of Technology), Todd. Stokes (Georgia Institute of Technology and Emory University), Carlton Dampier (Emory University), May D. Wang (Georgia Institute of Technology and Emory University)
- Detecting Fake Check-Ins in Location-based Social Networks Through Honeypot Venues
  Ke Zhang, Konstantinos Pelechrinis, Prashant Krishnamurthy (University of Pittsburgh)
- Leveraging Imperfections of Sensors for Fingerprinting Smartphones
  Sanorita Dey, Nirupam Roy, Wenyuan Xu, Srihari Nelakuditi (University of South Carolina)
- Participation Management for Mobile Crowdsensing
  Tingxin Yan, Jing Yang (University of Arkansas)
- Energy efficient semantic context model for managing privacy on smartphones
  Prajit Kumar Das, Dibyajyoti Ghosh, Anupam Joshi, Tim Finin (University of Maryland)
- A Group-based Security Protocol for Machine Type Communication in LTE-Advanced
  Hyoung-Kee Choi, Dae-Sung Choi (Sungkyunkwan University)
- The Importance of Timing in Mobile Personalization
  Chad Williams (Central Connecticut State University)
- Lifestreams Dashboard: an interactive visualization platform for mHealth data exploration
  Cheng-Kang Hsieh, Hongsuda Tangmunarunkit, Faisal Alquaddoomi, John Jenkins, Jinha Kang, Cameron Ketcham, Brent Longstaff, Joshua Selsky, Dallas Swendeman, Deborah Estrin, Nithya Ramanathan (University of California, Los Angeles)
- Mitigating Distractions from Smartphones
  Connor Bain, Sanorita Dey, Srihari Nelakuditi (University of South Carolina), Romit Roy Choudhury (Duke University)
- Vehicular Inter-Networking via Named Data
  Giulio Grassi, Davide Pesavento, Lucas Wang, Giovanni Pau (UCLA), Rama Vuyyuru, Ryuji Wakikawa (Toyota ITC), Lixia Zhang (UCLA)

Message from the Chairs

ACM HotMobile 2013 marks the fourteenth Workshop on Mobile Computing Systems and Applications. This year, it will be held in Jekyll Island, Georgia – moving to the opposite US coast from our 2012 workshop in La Jolla, CA. ACM HotMobile is a strong, peer-reviewed forum providing insight and debate into ground-breaking areas in mobile computing.

The workshop submission count was slightly lower compared to last year – 54 submissions this year and 68 last year. The program committee accepted 17, resulting in a higher acceptance rate of 31%. We accepted a number of poster and demonstration submissions. We will have several supporting events throughout the workshop to complement our rich technical program of six paper sessions. Our opening keynote is titled "Wearable Computing: Through the Looking Glass" from Thad Starner, a pioneer in the field of wearable computing from GeorgiaTech. We have a panel on "Mobile Systems and the Developing World", chaired by Gaetano Borriello of University of Washington.

In keeping with our deliberately small workshop theme, we will have a number of opportunities for attendees to engage with each other. At the end of every paper session, the session chair will bring up all the presenters and facilitate a deeper discussion on the topic covered by that session. We will have a poster and demonstration session, allowing PhD students to present and discuss their research with the goal of obtaining early critical feedback from an audience that has a wide perspective on the challenges of mobile computing. We will also have socializing opportunities, including a banquet, an ice cream social, and several breaks.

New this year, we are running a student travel grant program. We received 33 applicants for travel support, which is an astonishingly high number considering that attendance at HotMobile is around 80. We used a number of criteria to pick the 11 awardees, including academic merit, diversity, ability of the advisor or department to fund travel, relevance of student's work to the workshop, likelihood of student's impact on SIGMOBILE, and prior volunteer efforts in SIGMOBILE. These students will also have visible volunteer roles at the workshop, including writing a workshop report for ACM MC2R, and helping with the registration desk. Many thanks go to the ACM SIGMOBILE chair, Roy Want, for supporting this experiment and we hope it is a resounding success. Through a combination of generous support from industry and careful venue selection and contract negotiation, we were able to bring early registration fees down to a historical low of $175 for students and $290 for ACM professional members.

As always, these events are only successful because of the valuable personal time that volunteers are prepared to give to our research community. We thank the 21 members of our program committee for their dedication in reviewing submissions, and the steering committee for their guidance. We thank all the chairs providing logistical support with publicity, student grants and volunteering, posters and demos, and website updates. Most of all, we thank all the authors that submitted their work to make this a valuable forum for sharing and debating early research results.

Sharad Agarwal, General Chair
Alexander Varshavsky, Program Chair

Feb 2013

Program Committee

Rajesh Krishna Balan, *Singapore Management University, SG*
Suman Banerjee, *University of Wisconsin-Madison, US*
Nina Bhatti, *HP Labs, US*
Gaetano Borriello, *University of Washington, US*
Ramon Caceres, *AT&T Labs, US*
David Chu, *Microsoft Research, US*
Landon Cox, *Duke University, US*
Nigel Davies, *Lancaster University, UK*
Maria Ebling, *IBM, US*
Roxana Geambasu, *Columbia University, US*
Marco Gruteser, *WINLAB, Rutgers University, US*
Jason Hong, *Carnegie Mellon University, US*
Anthony LaMarca, *Intel, US*
Marc Langheinrich, *USI, Switzerland*
Stefan Saroiu, *Microsoft Research, US*
Mahadev Satyanarayanan, *Carnegie Mellon University, US*
James Scott, *Microsoft Research, UK*
Srini Seshan, *Carnegie Mellon University, US*
Roy Want, *Google, US*
Matt Welsh, *Google, US*
Lin Zhong, *Rice University, US*

Steering Committee

Nigel Davies, *Lancaster University, UK* **(Chair)**
Rajesh Krishna Balan, *Singapore Management University, SG*
Nina Bhatti, *HP Labs, US*
Ramón Cáceres, *AT&T Labs, US*
Landon Cox, *Duke University, US*
Angela Dalton, *JHU Applied Physics Lab, US*
Maria Ebling, *IBM Research, US*
Jason Hong, *Carnegie Mellon University, US*
Anthony LaMarca, *Intel Labs, US*
Eyal de Lara, *University of Toronto, Canada*
April Slayden Mitchell, *HP Labs, US*
Mahadev Satyanarayanan, *Carnegie Mellon University, US*
Roy Want, *Google, US*

Splitting the Bill for Mobile Data with SIMlets

Himanshu Raj, Stefan Saroiu, Alec Wolman, Jitendra Padhye
Microsoft Research

Abstract: The scarcity of mobile broadband spectrum is a problem hurting all stakeholders in the mobile landscape – mobile operators (MOs), content providers, and mobile users. Building additional capacity is expensive, and MOs are reluctant to make such investments without a clear way of recouping their costs. This paper presents the idea of *split billing*: allowing content providers to pay for the traffic generated by mobile users visiting their websites or using their services. This creates an additional revenue stream for MOs and builds more pressure for updating their networks' capacities. End users also benefit because they can afford more expensive data plans and enjoy new applications and scenarios that make use of faster mobile networks.

To implement split billing securely on a mobile platform, we develop the *SIMlet*, a new trustworthy computing abstraction. A SIMlet can be bound to a network socket to monitor and account all the traffic exchanged over the network socket. SIMlets provide trustworthy proofs of a device's mobile traffic, and such proofs can be redeemed at a content provider involved in split billing.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Wireless communication

General Terms

Design

Keywords

Mobile Computing, Traffic Metering, Traffic Billing, Broadband Wireless, 3G, 4G, ARM TrustZone, Split Billing, SIMlet

1. INTRODUCTION

The scarcity of spectrum for mobile broadband networks is hurting all stakeholders in the mobile landscape – users, content providers, and the mobile operators (MOs). As users migrate more of their computing tasks to smartphones and tablets using 3G/4G networks, users will increasingly feel constrained by the low capacity of today's mobile broadband networks. Content providers have little choice but to create "mobile-only" versions of their content to accommodate the lack of bandwidth reaching smartphones and tablets on the go. Finally, MOs are reluctant to make costly investments in upgrading their infrastructure. While MOs have access to a variety of approaches to improve their networks' capacity, all such approaches come with significant costs. Indeed, whether MOs purchase more spectrum, increase spatial reuse by deploying a higher density of cell towers, or deploy femtocells, MOs need strong incentives to invest billions of dollars in their infrastructure.

The current situation resembles a "chicken and egg" problem. MOs have little incentive to make expensive investments into their infrastructure as long as it is not clear how their costs will be recouped. Users are stuck with moderately expensive data plans, and are unwilling to pay more in the absence of compelling content and applications that need faster mobile broadband networks. Finally, content providers and application developers cannot "force" MOs to update their networks; instead, they alter their applications and services to fit current bandwidth rates and "spotty" reception.

There are only a few examples of content providers who have managed to overcome this situation. One example is Amazon's Kindle devices, which are sold with no need for customers to purchase data plans. Instead, Amazon bought data in bulk directly from AT&T. Amazon will recoup the cost of the 3G data as long as customers buy books from Amazon, and watch ads on their Kindles[1], all activities that consume relatively little bandwidth. Other examples are *0.facebook.com* [4], where Facebook offers free access to their site for all users of certain MOs, and Globe Telecom's *Free Zone*, a partnership between an MO in the Phillipines and Google [5]. Unfortunately, all such solutions face two challenges. First, they are heavy-handed because content providers and application developers must now sign deals with MOs; although Amazon can negotiate a deal with AT&T, it is much more difficult for the average app developer to create such a deal. The second challenge is that it is also expensive for MOs to negotiate individual deals with a large number of content providers. Such an approach to overcoming the "chicken and egg" problem does not scale.

This paper explores a different approach to overcoming this problem: *split billing*, a mechanism that allows content providers to subsidize or share the cost of 3G data consumed by their content and services. Split billing lowers the barriers between MOs, content providers, and users because it lets content providers to offer to pay for bandwidth on behalf of customers. This creates additional revenue streams for MOs, incentivizing them to upgrade their networks' capacities. End users also benefit because new scenarios are now possible, such as:

[1] Amazon offers cheaper versions of their devices that require users to watch ads. Ad-free Kindles are also available, but are more expensive.

HotMobile'13, February 26–27, 2013, Jekyll Island, Georgia, USA.

- Bandwidth-intensive websites such as Netflix or Hulu can enable mobile users to visit them and not worry about exceeding their bandwidth caps.

- Users can have a single phone for both work and personal use, and bill the network costs for their enterprise VPN traffic and other work-related applications directly to their employer.

- Advertisers can offer much richer ads (e.g., short video clips) to mobile users without having to worry about affecting their data plan quotas.

- Mobile phone users can run applications, such as peer-to-peer applications, where the traffic generated by the application is done on behalf of another user or device.

- Parents can encourage teenagers to run continuous monitoring applications to let them know where they are.

All these usage scenarios share a common restriction – a certain type of traffic meant for a certain destination or for use at a particular time should not be counted against the default quota.

Implementing split billing requires content providers and app developers to track the amount of 3G traffic their users consume. The system must have a high degree of security otherwise users might cheat in an attempt to qualify for savings on their 3G bill, by falsely claiming to have used 3G to access a site when in fact they used Wi-Fi. One possible implementation is to track users on the server-side and classify the type of network they are using (as 3G or Wi-Fi) based on the client's IP address. It is challenging to accurately infer which IP addresses are 3G versus Wi-Fi unless MOs cooperate with content providers and app developers. Section 4 describes the challenges of server-side split billing in more depth.

Instead, we chose an implementation that does not require cooperation from MOs: tracking 3G usage on the client-side. A client-side solution is more readily deployable because it does not require engagement with MOs. After a client-side approach has demonstrated the benefits of split billing, MOs will become more willing to take the necessary steps to offer split billing support to the masses of content providers and app developers. For example, MOs could offer a secure cloud-side service that tells app developers whether a given IP address is connected through their mobile broadband network. The app developer would use such a service to offer split billing inside their apps. Such a service must offer a high-degree of trustworthiness, otherwise mobile users could misrepresent their Wi-Fi traffic as mobile broadband traffic to the application's split billing component.

To implement client-side split billing, we develop an abstraction called a *SIMlet*. SIMlets are simple packet filters that associate individual traffic with a particular billing account. SIMlets must remain secure in the face of OS vulnerabilities, and thus they run in a strongly isolated runtime that leverages trusted computing hardware for mobile devices. Some of today's mobile phones already support dual SIM cards. SIMlets can be thought of as equivalent to being able to insert multiple SIM cards into a phone, along with a set of rules that indicate which particular SIM card a network flow should use.

Using SIMlets, we started the design and implementation of a fine-grained networking billing system for mobile devices where client applications or individual websites can create billing accounts and specify classes of traffic to be associated to a particular billing account. For each account, the system can provide trustworthy proofs of how much data applications have consumed. Users could redeem such proofs to MOs or content providers in exchange for rewards, rebates, credits, or even cash.

For our implementation, we use the Berkeley Packet Filter (BPF) to support the SIMlet abstraction, and we use the ARM TrustZone feature built into modern ARM System-on-Chip's (SoCs) to provide a trusted execution environment for SIMlets. Using TrustZone enables our system to perform trustworthy metering of which traffic is assigned to which billing account even if malware infects the OS running on the smartphone or tablet.

2. COST ANALYSIS

3G/4G spectrum crunch stems from the limited spectrum and long-range nature of the wireless technology. A single cell tower today handles 100's or even 1000's of individual subscribers at distances of up to tens of miles [13]. This is in sharp contrast to Wi-Fi where a one AP rarely handles more than tens of subscribers. With the unprecedented growth of smartphone and tablet usage, there simply is not enough network capacity to address the emerging demand. While we have seen a steady increase in the data rates supported by mobile broadband networks, the effective throughput for applications is substantially lower than the advertised rates [9], and we are already witnessing the effects of network congestion, with many users complaining of slow networks.

To address these capacity issues, network providers are considering a few options: 1) purchasing additional wireless spectrum, 2) increasing spatial reuse by deploying additional celltowers, and 3) increasing spatial reuse by deploying femtocells. While each of these options has the potential to improve the current capacity crunch, there is a common factor hindering all these options. They are each very expensive, and they come with significant deployment challenges. For example, increasing celltower density requires leasing or buying more physical locations, obtaining permits, and connecting all these sites, which increases fixed network costs and complicates operations.

For split billing to be viable, the customer base must be attractive to content providers and app developers. Content providers and app developers must be able to recoup the split billing costs and even sell more services to customers. For example, a movie download provider could offer free movie previews to entice its customers to purchase more movies. A mobile game developer might offer free 3G for its games as long as customers keep making in-app purchases. Split billing is attractive as long as content providers and app developers can target customers with the means to pay for additional content and services.

The remainder of this section describes a cost analysis of the feasibility of split billing. We combine three datasets to show that today's data plans come with strict quotas that provide little data to their users. The second high-level finding is that some well-developed countries offer worse data plans (by both cost and quantity of data) than some less-developed countries. This suggests that split billing could be effective in attracting customers more likely to spend on additional content and services.

2.1 Data Sources

The data plans used in our analysis come from a dataset of prices of mobile broadband released by Google[2]. Table 1 shows a few high-level statistics of this dataset. We also breakdown some of our analysis by countries and their GDP; the GDP data comes from

[2] http://policybythenumbers.blogspot.gr/2012/08/international-broadband-pricing-study.html

	Mobile Broadband Pricing (Google dataset)
Number of Plans Examined	2154
Number of Countries	106
Types of Mobile Broadband	2G, 2.5G, 3G, 4G, CDMA, EDGE, EVDO, HDSPA, HSDPA, HSPA, LTE
Number of Unlimited Plans	142 (4.9%)

Table 1: High-level statistics of the mobile broadband pricing data released by Google.

the World Bank[3]. Finally, we use estimates of the today's 3G/4G bandwidths collected by a very recent paper [16].

2.2 Today's Data Plans Offer Little Data

The pricing dataset revealed by Google shows that most data plans have very strict quotas. On the left, Figure 1 illustrates the distribution of monthly quotas. Almost 85% of all plans offer less than 10GB of data a month, and 36% offer less than 1GB a month. Even worse, while unlimited data plans which were common a few years ago have all but disappeared. In fact, we found less than 5% of plans to offer unlimited data (these plans were removed from the data plotted in Figure 1). Unfortunately, we cannot investigate the popularity of each data plan because the pricing dataset does not include the number of subscribers for each plan.

To better display how little data these plans offer, on the right, Figure 1 illustrates how many hours of downloads (or uploads) a 10GB plan offers given current 3G/4G bandwidth found in three major US cities: New York, Los Angeles, and Chicago. We use recent measurements of average download and upload speeds collected by [16]. In NY alone, a 10GB monthly data plan would be fully exhausted in less than 7 hours. Since uploads are slower, a 10GB plan could sustain about 12 hours of uploads a month. These findings show that today's plans offer little data to their users.

2.3 The Viability of Split Billing

To examine this issue in more depth, we perform the following experiment. We classify the countries present in the Google dataset in two ways: by their GDP per-capita in US$ and by the per gigabyte cost of their cheapest data plan. If some of the "richest" customers have access to inexpensive data plans then split billing is less attractive to them. Conversely, if data plans are very expensive only to those customers living in countries with low GDPs, these customers are also less likely to afford additional content and services.

Table 2 lists the top 10 "richest" countries in each category: by GDP in US$ and by per-GB cost. The two datasets are completely disjoint; seven of the top 10 "richest" countries by per-GB cost are located in Africa, a continent that has no country in the top 10 "richest" by GDP. Note that we only consider the countries listed in the Google dataset; for example, although Luxembourg has the highest per-capita GDP in the world, it is not listed in Table 2 because the Google dataset does not include any data plans from Luxembourg.

Even worse, unlimited plans are equally unavailable to customers in countries with high GDP as well as low GDP. Out of the top 20 countries with highest GDP, only seven of them have access to unlimited data plans. In contrast, six countries out of the bottom 20 also have access to unlimited data. All these results show that high-GDP customers do not benefit from cheaper plans and more data. All these findings suggest that split billing could be used effectively to attract this class of customers.

[3] http://data.worldbank.org/indicator/NY.GDP.PCAP.CD

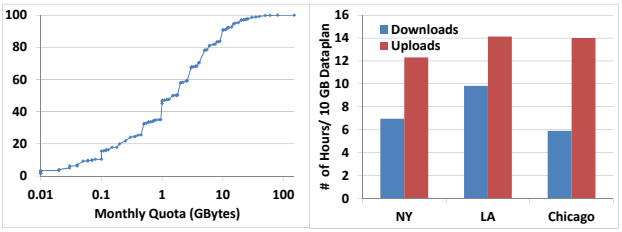


Figure 1: The distribution of monthly quotas for mobile broadband plans (left), and the number of hours of connectivity for monthly plans in three major US cities (right).

3. OUR SPLIT BILLING DESIGN

Split billing is the division of a customer's data plan bill into multiple parts. Our design deliberately avoids relying on MOs to make changes inside their networks. This design choice leads to a much quicker path to deploying split billing in the wild. After the benefits of split billing have been demonstrated, MOs can then offer different mechanisms that implement split billing support inside their network. Therefore, we start with the following four design goals:

1. *No changes inside the network.* Our system should be implemented by making changes to the endpoint mobile devices only. The design originated from our desire of quick prototyping. Requiring changes to the mobile network would drastically raise deployment costs and challenges.

2. *The security of split billing should be comparable to the security of SIM cards today*. It should be very difficult for users to bypass our security and charge for "bogus" traffic. Although impossible to offer "perfect security" (e.g., SIM cards can be lost and stolen, or subject to sophisticated physical attacks [7]), we aim to maintain similar levels of security as with today's SIM cards.

3. *Offer a form of accountability.* Anyone, whether a user, content provider, or mobile operator, should be able to verify who is responsible for the data plan charges.

4. *Provide adequate performance.* Any performance overhead should be negligible to users.

3.1 High-Level System Overview

In our system, each mobile device manages a set of rules on how to bill the 3G/4G data. For example, data consumed by visiting Netflix could be billed to the user's Netflix account, whereas data consumed over an enterprise VPN connection could be billed to the user's employer. Split billing runs strongly isolated from the rest of the system in such a way that any security compromise (including an OS compromise) will not compromise network metering. Periodically, our system produces a bill that tallies the 3G traffic consumed against each of the rules entered in the system.

We rely on the mobile device's trusted computing features (ARM TrustZone [3]) to isolate the split billing code from the rest of the running system and to produce an attestation for each bill (e.g., using a mechanism similar to remote attestation provided by TPMs [19]). ARM TrustZone enables this approach: it provides two runtime environments called the normal world and the secure world, with hardware support for isolation. This ensures that secure world memory is isolated from the normal world. The OS and all applications run in the normal world, and only the trusted components that implement split billing run in the secure world. A

	GDP Per-Capita	Cost per GByte
1.	Switzerland	Algeria
2.	Australia	Papua New Guinea
3.	Denmark	Cameroon
4.	Sweden	Iraq
5.	Canada	Angola
6.	Holland	Haiti
7.	Austria	Zimbabwe
8.	Finland	Chad
9.	United States	Mali
10.	Belgium	Sierra Leone

Table 2: Top 10 countries by GDP per-capita and cost per gigabyte.

more in-depth description of the ARM TrustZone technology can be found in [3].

This high-level description raises two challenges. First, what is a good abstraction for encapsulating the set of rules on a device that dictates how to split the the user's 3G bill? Second, how can we implement this abstraction securely by leveraging the TrustZone found on commodity ARM-based mobile devices?

3.2 Implementing Split Billing with SIMlets

To implement split billing, we develop a new abstraction, called a *SIMlet*. A SIMlet corresponds to a billing account, is issued by a content provider, and is destined to a smartphone user. A SIMlet also includes a *rule* that dictates the conditions under which 3G data is billed against the SIMlet. For example, Netflix could issue a SIMlet that specifies all 3G data to or from *netflix.com* is to be paid for by Netflix.

The issuer of a SIMlet is a content provider identified by its domain name and the recipient is a smartphone user identified by their phone number. A SIMlet has a start and an end date and can be used only during this time. We use Berkeley Packet Filter (BPF) syntax to specify the SIMlet's policy, although SIMlets could use a more expressive policy language if needed. Finally, SIMlets are signed by their issuers to prevent their modification.

3.3 Trusted SIMlet Manager

Each smartphone has a trusted SIMlet manager whose role is to request and store SIMlets. The SIMlet manager runs inside the TrustZone secure world and is therefore isolated from the smarthone's OS and applications. Any application can ask the SIMlet manager to request a SIMlet from a remote website. The SIMlet manager contacts the remote content provider, authenticates to it, and requests the SIMlet. The manager then stores the SIMlet locally and signals to the application whether the SIMlet request was successful or not. We envision two ways in which the trusted SIMlet manager can authenticate to a remote website, either using primitives provided by the physical SIM card or using TPM-like remote attestations.

Once it issues a SIMlet, a remote content provider commits to pay for a portion of the user's traffic. The provider is free to share this information with the MO if it wishes to do so. This could help with streamlining the customer's payment process, or it could form the basis for further negotiations between the MO and the content provider with respect to bandwidth prices.

3.4 Trusted Metering with SIMlets

Our system runs a trusted meter in the secure world whose role is to match network traffic against the appropriate SIMlets. For this, an application explicitly requests that the trusted meter bill its traffic against a particular SIMlet. The trusted meter instantiates a matching rule in a forwarding table that will match the application's traffic and meter it appropriately. To implement this approach, we use sockets and packet filters.

```
// find a simlet that can be used for our destination
IPSimlet ips = IpSimlet.DefaultSimlet;
foreach (IPSimlet tmp in SimletStorage) {
  if(tmp.Issuer.Contains("hulu.com")) {
    ips = tmp;
  }
}

// create socket
Socket s = new Socket(AddressFamily.InterNetwork,
                      SocketType.Stream, ProtocolType.TCP);
IPHostEntry ipHostInfo = Dns.Resolve("www.hulu.com");
IPAddress ipAddress = ipHostInfo.AddressList[0];
IPEndPoint ipe = new IPEndPoint(ipAddress, 80);

// bind the simlet to the socket
s.BindSimlet(ips);

//connect
try {
  s.connect()
} catch(SimletException sime) {
  Console.WriteLine("SimletException: {0}", sime.ToString());
} catch(SocketException socke) {
  Console.WriteLine("SocketException: {0}", socke.ToString());
} catch(Exception e) {
  Console.WriteLine("Unexpected exception: {0}", e.ToString());
}
```

Figure 2: C# code fragment for creating a socket and binding a SIMlet to it.

When opening a socket, an application requests a SIMlet that is then bound to the socket. This binding operation allows applications to decide which SIMlets they want to use, which ensures that the user has control of this process. Once a socket has been bound to a SIMlet, the smartphone OS will send all outgoing network traffic from the socket buffer to the TrustZone secure world. Once in the secure world, the SIMlet packet filter rules are evaluated for billing, and then the data is handed to the network stack. For incoming traffic, the network stack decides which packets are destined to which sockets, and then checks if the socket has a SIMlet bound to it. If so, the SIMlet is evaluated, and then the packets are handed off from the secure world back to the socket in the normal world. Figure 2 shows a C# example of how to create a TCP socket and bind a SIMlet from *hulu.com* to it.

4. SERVER-SIDE SPLIT BILLING

Another design alternative is to implement split billing on the server-side. The advantage of such a design is that it does not require any modifications to clients, and there is no need to leverage trusted computing for strong isolation. Servers are trustworthy because they are already under the control of the content provider, unlike mobile devices. However, a server-side architecture must solve the following two challenges.

First, content providers need to distinguish what type of network link (e.g. Wi-Fi vs. 3G) their customers use to connect to their services. This step must be done securely; a process relying on customers to *notify* servers when using 3G versus Wi-Fi could be abused. To qualify for rewards and subsidies, customers could cheat and claim they are using 3G when actually using Wi-Fi.

Instead, servers must classify IP addresses as 3G or Wi-Fi without the cooperation of clients. Since MOs control the space of IP addresses allocated to their 3G clients, one possibility is to ask the MOs to help with IP classification. Unfortunately, MOs do not currently advertise what IP address ranges they use for 3G, and, in our experience, they are reluctant to share this knowledge with app developers and content providers at large.

Another possibility is to use network measurements rather than

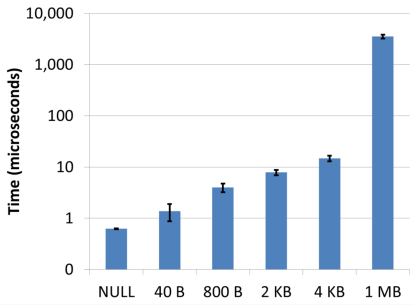


Figure 3: Overhead of RPC from normal world into secure one as a function of the size of RPC data.

MO cooperation to build catalogs of 3G IP addresses. Such a map can be constructed by combining a variety of techniques that gather information about IP addresses (e.g., reverse DNS lookup, autonomous system (AS) lookup, traceroute) and use various registries (e.g. RouteView [1], whois) to lookup information [17].

The second challenge is traffic counting. Server infrastructure is often geographically distributed, and sometimes these servers can be in different administrative domains under someone else's control, such as the case with content delivery networks (CDNs), or hosted servers (e.g., Microsoft's Azure or Amazon's AWS). It is often difficult to count traffic accurately across different servers placed in different organizations.

The major advantage of the server-side alternate design is that it requires no software deployment on the client, and thus there is no need for client-side trusted computing hardware. However, in the absence of cooperation from the MO, we think the problem of accurate IP address classification may be quite challenging to overcome. If IP classification is inaccurate, attackers will find ways to exploit it and cheat the split billing scheme.

5. PRELIMINARY EVALUATION

A key performance concern is the context-switching cost of transitioning from the normal world to secure world and back. To understand these costs, we performed a series of experiments in which the normal world executes a Remote Procedure Call (RPC) that transitions to the secure world, copies the RPC's arguments, and returns back a result. Such an experiment is a rough approximation of the split billing overhead as we must transition into the secure world each time data has to be metered.

Because all data sent through a socket with a SIMlet bound to it must be copied into the secure world, in our experiments we look at the cost of sending data of various sizes (40 bytes, 800 bytes, 2 KB, 4 KB, and 1 MB) in addition to performing null RPCs (i.e., no data copies). Each experiment is repeated 100 times and we present the average result and the standard deviation. Our experiments were done on an NVidia Harmony board with the Tegra 250 SoC. This SoC contains a dual core ARM Cortex A9 CPU running at 1 GHz with 1 GB of RAM.

Our results show that the cost of such transitions is very low. A null RPC call takes on average 624 nanoseconds with a standard deviation of 11.6 ns. Figure 3 shows the cost of an RPC call as a function of the size of the data copies. We find the cost of RPCs meets the performance needs of split billing. Even when copying 1MB of data to the secure world, the transitioning overhead due to metering is about 3.5 milliseconds.

6. RELATED WORK

A recent paper suggests that MOs should use time-dependent pricing of mobile data to manage their demand [8]. Similar to our work, this work's motivation stems from the huge growth in the demand for mobile data and the lack of capacity. However, charging more for data (which is what time-dependent pricing must do to reduce demand) makes mobile data plans even worse than they currently are. Instead, this work takes a different approach by engaging content providers and app developers in sharing the costs of mobile data.

Our system design borrows ideas from trusted computing hardware primitives to protect integrity and confidentiality for code and data [15, 12, 20, 11, 14]. Most of this previous work leverages the Trusted Platform Module (TPM) chips found on x86 hardware platforms [15, 12, 11]; few projects investigate the use of ARM's trusted computing primitives (ARM TrustZone) [10, 14, 20].

Our work has many parallels to the network neutrality debate and Section 7 examines this issue in more depth. Several previous projects have built and deployed techniques for quantifying the prevalence of traffic shaping on the Internet [18, 6], and recent work investigates the presence of traffic shaping in 3G networks [2].

7. DISCUSSION

This section presents three issues facing our implementation and the reasons why we remain optimistic that our system can overcome them. We do not claim to understand all issues our system will face in practice, nor that our way of addressing them will be the most effective. Our main goal is to obtain feedback at an early stage.

Split billing will increase the diversity of bandwidth pricing models which will in fact increase the incentives for traffic discrimination. It is true that split billing does not enforce the elimination of traffic discrimination. Large content providers can still negotiate preferential rates for their customer's traffic. However, one of the benefits provided by split billing is that it makes the network and traffic pricing more transparent. We believe that small content providers can only gain from increased transparency into data pricing and from access to a means for subsidizing their customers' bandwidth costs. Today's lack of transparency only hurts them, whereas our system democratizes access to billing agreements.

Mobile operators (MOs) fear of commoditization. Our system provides a simple way for content producers and users to enter agreements on who pays for bandwidth. One could argue that such agreements will make MOs feel that they have less control over the price of bandwidth and increase their fear of commoditization. Although we acknowledge that split billing might produce such an impression to MOs at first, we think MOs would embrace our system because it gives any content provider a simple way to pay for the bandwidth of its customers. Ultimately, MOs carry the traffic and thus can exert control over bandwidth prices. Also, split billing increases the pool of payees by including content providers (or any third-party) who may be willing to pay higher prices than consumers can tolerate.

Running a part of the billing system on a mobile device is too much of a security risk. This concern is serious – a security compromise of the TrustZone components of our system would allow an attacker to bill arbitrary 3G data traffic to any SIMlet. One way to mitigate this threat (other than the obvious way of strengthening the security of our system) is to build a way for content providers (or MOs) to audit the bills. For example, a content provider could also meter a customer's traffic on the server side and check whether

the bill presented by the customer matches the amount of traffic recorded at the server. Such audits do not have to run continuously and check all customers; instead, spot-checks would be able to determine whether our system provides adequate security.

8. CONCLUSIONS

This paper puts forward the idea of split billing: allowing content providers and app developers to pay for the traffic generated by mobile users when visiting their websites or using their services. We present a preliminary design of split billing on the client-side, without requiring any cooperation from the mobile operators. We believe that a client-side design offers a quick way to deploy split billing in the wild because it makes no assumptions about the network and takes no dependencies on the mobile operators.

To implement split billing securely on the client-side, we introduce a new trustworthy computing abstraction, called a SIMlet. We present a simple C# API on how SIMlets can be used by mobile applications. Finally, our evaluation shows that the overhead of copying data to and from the ARM TrustZone, an operation necessary for implementing SIMlets in a trustworthy manner, is low.

Acknowledgments

We would like to thank our shepherd, Suman Banerjee, and the anonymous reviewers for their valuable comments and feedback.

9. REFERENCES

[1] University of Oregon Route Views Project. http://www.routeviews.org/.

[2] WindRider: A Mobile Network Neutrality Monitoring System. http://www.cs.northwestern.edu/~ict992/mobile.htm.

[3] ARM Security Technology – Building a Secure System using TrustZone Technology. ARM Technical White Paper, 2005-2009.

[4] Fast and Free Facebook Mobile Access with *0.facebook.com*. http://www.facebook.com/blog/blog.php?post=391295167130, 2010.

[5] The Phillipines gets Facebook Zero-style free mobile access to Google services via Globe Telecom. http://thenextweb.com/mobile/2012/11/08/google-and-globe-telecom-launch-freezone-for-mobile-access-to-web-and-select-google-sites/, 2012.

[6] M. Dischinger, M. Marcon, S. Guha, K. P. Gummadi, R. Mahajan, and S. Saroiu. Glasnost: Enabling End Users to Detect Traffic Differentiation. In *Proc. of NSDI*, 2010.

[7] C. Gui, H. J. Wang, and W. Zhu. Smart-Phone Attacks and Defenses. In *Proc. of HotNets*, 2004.

[8] S. Ha, S. Sen, C. Joe-Wong, Y. Im, and M. Chiang. TUBE: Time-Dependent Pricing for Mobile Data. In *Proc. of Sigcomm*, 2012.

[9] J. Huang, Q. Xu, B. Tiwana, Z. M. Mao, M. Zhang, and P. Bahl. Anatomizing Application Performance Differences on Smartphones. In *Proc. of MobiSys*, 2010.

[10] H. Liu, S. Saroiu, A. Wolman, and H. Raj. Software Abstractions for Trusted Sensors. In *Proc. of MobiSys*, 2012.

[11] J. M. McCune, Y. Li, N. Qu, Z. Zhou, A. Datta, V. Gligor, and A. Perrig. TrustVisor: Efficient TCB Reduction and Attestation. In *Proc. of IEEE Symposium on Security and Privacy*, 2010.

[12] J. M. McCune, B. Parno, A. Perrig, M. K. Reiter, and H. Isozaki. Flicker: An Execution Infrastructure for TCB Minimization. In *Proc. of Eurosys*, 2008.

[13] RYSAVY Research. Mobile Broadband Capacity Constraints And the Need for Optimization. http://www.rysavy.com/Articles/2010_02_Rysavy_Mobile_Broadband_Capacity_Constraints.pdf, 2010.

[14] N. Santos, H. Raj, S. Saroiu, and A. Wolman. Trusted Language Runtime (TLR): Enabling Trusted Applications on Smartphones. In *Proc. of Hotmobile*, 2011.

[15] A. Seshadri, M. Luk, N. Qu, and A. Perrig. SecVisor: A Tiny Hypervisor to Provide Lifetime Kernel Code Integrity for Commodity OSes. In *Proc. of SOSP*, 2007.

[16] J. Sommers and P. Barford. Cell vs. WiFi: On the Performance of Metro Area Mobile Connections. In *Proc. of IMC*, 2012.

[17] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP Topologies with Rocketfuel. In *Proc. of SIGCOMM*, 2002.

[18] M. B. Tariq, M. Motiwala, N. Feamster, and M. Ammar. Detecting Network Neutrality Violations with Causal Inference. In *Proc. of CoNEXT*, 2009.

[19] Trusted Computing Group. PC client specific TPM interface specification (TIS), 2005.

[20] J. Winter. Trusted Computing Building Blocks for Embedded Linux-based ARM TrustZone Platforms. In *Proc. of STC*, 2008.

Towards Accurate Accounting of Cellular Data for TCP Retransmission

Younghwan Go, Denis Foo Kune†, Shinae Woo, KyoungSoo Park, and Yongdae Kim

KAIST University of Massachusetts Amherst†

ABSTRACT

The current architecture supporting data services to mobile devices is built below the network layer (IP) and users receive the payload at the application layer. Between them is the transport layer that can cause data consumption inflation due to the retransmission mechanism that provides reliable delivery. In this paper, we examine the accounting policies of five large cellular ISPs in the U.S. and South Korea. We look at their policies regarding the transport layer reliability mechanism with TCP's retransmission and show that the current implementation of accounting policies either fails to meet the billing fairness or is vulnerable to charge evasions. Three of the ISPs surveyed charge for all IP packets regardless of retransmission, allowing attackers to inflate a victim's bill by intentionally retransmitting packets. The other two ISPs deduct the retransmitted amount from the user's bill thus allowing tunneling through TCP retransmissions. We show that a "free-riding" attack is viable with these ISPs and discuss some of the mitigation techniques.

Categories and Subject Descriptors

C.2.0 [**General**]: Security and protection; C.2.1 [**Network Architecture and Design**]: Packet-switching networks; C.2.6 [**Internetworking**]: Standards

Keywords

Cellular Networks, TCP Retransmission, Accounting, Charging

1. INTRODUCTION

Cellular 3G/4G data traffic is rapidly increasing. The volume is predicted to reach 10.8 Exabytes per month in 2016, which is an 18-fold increase from that of 2011 [1]. The number of cellular network users has already reached 1.2 billion worldwide [2], and it is estimated that 85% of the world population will subscribe to the cellular network service by 2017 [3].

Given the increasing demand in the cellular traffic, accurate accounting of the traffic usage becomes all the more important. Most cellular ISPs adopt the pay-per-usage charging model for cellular Internet access. Subscribers typically buy a monthly usage plan (e.g., 3 GB per month) and the ISPs enforce it by byte-level accounting of the consumed IP packets. However, this approach presents an important policy decision for the TCP traffic. ISPs now need to decide whether they account for retransmitted TCP packets or not. If the ISPs reflect the retransmitted packets into the bill, it may be unfair to the users especially when the packet delay variance or losses are due to a poorly-provisioned infrastructure. In our measurement at one ISP in South Korea, we observed some flows with up to 93% of the packets being retransmitted due to packet loss. What is worse is that the blind accounting policy can be easily abused by malicious attackers that try to inflate the cellular traffic usage for a specific user or even for all users from a specific ISP. The natural alternative is to remove the retransmitted packets from the bill, but accounting becomes expensive since it has to manage every TCP flow for each subscriber.

In this work, we present the implications of byte-level accounting policies in the cellular traffic for TCP retransmission. The root cause of the problem lies in that the majority of the mobile data traffic flows over TCP [4–7], which ensures the flow-level reliability by transparently retransmitting the lost packets [8]. However, the ISPs account for each IP packet, which sometimes creates a disparity in what users perceive and what the infrastructure provides.

To better understand the current practice, we examine the accounting policies for TCP retransmission with five large cellular ISPs in the U.S. and in South Korea. Surprisingly, we find that the accounting policies vary between ISPs, and that even the ISPs in the same country have different policies. Our measurements reveal that three ISPs (two in the U.S. and one in South Korea) account for every packet regardless of TCP retransmission. We further confirm that the users in these ISPs can be the target of a usage-inflation attack that maliciously retransmits packets even if there are no packet loss. The remaining two ISPs (both in South Korea) intentionally remove the retransmitted amount from the usage statistics. However, we find that their implementation allows free data transfers if attackers tunnel their packets inside TCP retransmissions. This implies that the ISP accounting system checks only the TCP headers for retransmission and does not check the actual content of the payload; doing so could be expensive in terms of storage and computation to recall previous payload contents, and compare them to suspected retransmissions.

Our contributions in this paper are summarized as follows. First, we report that the current byte-level accounting for TCP retransmission fails to meet the fairness nor the correctness in billing. Blind accounting of every packet leads to unfair usage inflation if the retransmission happens due to infrastructure-induced congestion or degraded wireless links. Second, we show that the current practice of cellular traffic accounting is vulnerable to attacks that either inflate the usage or send the packets without being charged. Peng et.

ACM HotMobile'13, February 26–27, 2013, Jekyll Island, Georgia, USA.

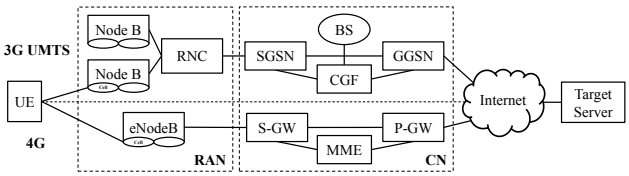


Figure 1: Overall architecture of 3G/4G cellular network

al. showed a similar attack exploiting a loophole in an ISP policy that blindly passes all packets on port 53 (DNS) at no charge [9,10]. While their work can be considered as "bugs" in the accounting policy, we believe that the accounting policy for TCP retransmissions is a *fundamental* problem tied to the basic mechanisms of the TCP layer. We argue that cellular ISPs should not count retransmitted bytes against the user's data plan, but they should also make sure that the retransmissions are legitimate to prevent abuse. Later in this paper, we discuss a few possible solutions that can be used to prevent (or mitigate) the vulnerability while maintaining a reasonable accounting load even for high-throughput networks.

2. BACKGROUND

In this section, we describe the basic architecture of 3G/4G cellular networks and their accounting process. We mainly focus on the Universal Mobile Telecommunications System (UMTS) [11] for 3G and Long Term Evolution (LTE) [12] for 4G. The architecture is based on a Packet-Switched (PS) domain, in which the data is transferred in packets [13,14]. Although we mainly focus on 3G, similar argument can be made for the 4G system as well.

2.1 3G/4G Accounting System Architecture

Figure 1 shows the overall architecture of UMTS/LTE cellular network. The User Equipment (UE, i.e. cellular devices such as smartphones, tablet PCs, etc.) communicates with a target server in the wired Internet by passing the packets through the UMTS, which consists of a Radio Access Network (RAN) and a Core Network (CN). The RAN is responsible for allowing wireless access to the UE and for providing a connection to its CN. Inside the RAN, Node B, a base station for transmitting and receiving data directly with the UE through an air interface, is controlled by a Radio Network Controller (RNC), which manages radio resources and UE mobility. In 4G, the RAN consists of only E-UTRAN Node B (eNodeB) without a RNC since eNodeB also has the control functionality embedded in it.

After passing through the RAN, the packets from a UE enter the General Packet Radio Service (GPRS) through Serving GPRS Support Node (SGSN), which is responsible for delivering packets to or from the UE within its service area. Then, the Gateway GPRS Support Node (GGSN) converts the GPRS packets coming from the SGSN into an appropriate Packet Data Protocol (PDP) format such as IP, and sends them out to an external data network such as the wired Internet where the target server is located. In 4G networks, the basic procedure is the same except for the fact that the SGSN is replaced with a Serving Gateway (S-GW), the GGSN is replaced with a packet data network gateway (P-GW), and the UE's mobility is handled by a mobility management entity (MME).

The cellular data accounting is carried out inside the CN in the form of a Charging Data Record (CDR), which includes the information necessary for billing such as the user identity, the session and the network elements, and services used to support a subscriber session. The CDR is generated by the serving nodes (SGSN, GGSN, S-GW, P-GW) and is forwarded via the Charging Gateway Function (CGF) with the charging information to the Billing System (BS). The CGF can be located anywhere: in an external interface, in every GSN, or in a particular GSN to serve other GSNs.

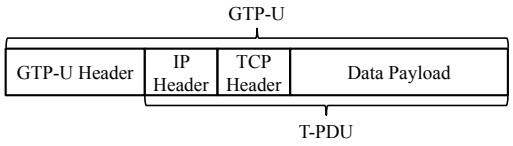


Figure 2: GPRS packet format inside the CN

2.2 3G Accounting Process

When a user establishes a connection with a target server to download some content, it triggers both GSNs to create their own CDRs (S-CDR, G-CDR) related to PDP contexts with the UE's unique charging ID (C-ID) to collect the charging information. The SGSN collects the charging information related with the radio network usage while the GGSN collects that of the external data network usage. The standard charging information collected by GSNs are the radio interface, usage duration, usage of the general packet-switched domain resources, source and destination IP addresses, usage of the external data networks, and the location of the UE.

While the UE downloads its requested content from the target server through the cellular network, the GSNs record the traffic volume arriving to the CN in the form of T-PDU (Figure 2). The T-PDU is the original IP packet received from either the UE or the target server, which is then converted in the CN to move around the GSNs. The T-PDUs are passed between GSN pairs via GTP-U tunnels by attaching the GTP-U header at the front [15]. The accounting process continues until the communication is completed and the UE tears down the connection. When the session is finished, the CDRs stored in the GSNs are forwarded to the BS via the CGF and are processed to calculate the total data volume consumed by the particular session. For byte-level accounting per user, most cellular ISPs account for entire IP packet sizes while their policies differ as to whether they include retransmitted TCP packets or not.

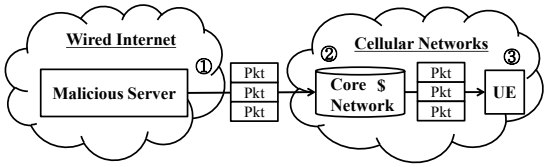
3. ACCOUNTING CHALLENGES IN TCP

In this section, we discuss the accounting issues in TCP-based Internet services in cellular networks. Since the majority of the cellular traffic is based on TCP, accounting of the TCP traffic directly affects the user bill. We first present the service provider's dilemma in accounting for TCP-level retransmission, and discuss the level of retransmission measured in real networks.

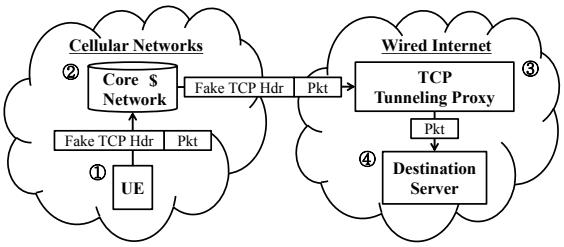
3.1 The Cellular Provider's Dilemma

From the cellular ISP's perspective, all headers and payloads from OSI layer 3 and above should be counted as well as retransmissions from layer 4 since they are consuming the cellular network's resources. However, packet retransmissions depend on the network conditions that are typically beyond the control of users. From the perspective of users, the useful data sits in the application layer and only the volume in the application layer should be counted. If the cellular service providers choose the latter, retransmission packets will be treated as a simple overhead. One such implementation is to bypass all retransmission packets whose TCP sequence numbers are older than the next expected sequence number. While this approach is efficient in that it checks only the TCP headers, we find that a naïve implementation is dangerous in practice.

This situation opens up possible attacks on either side of the dilemma as shown in Figure 3. If the provider accounts for the



(a) Usage-inflation attack by a malicious server: 1) A malicious server sends retransmission packets to the client user equipment even if there is no timeout or 3 duplicate ACKs. 2) The core network accounts for all retransmitted packets. 3) The client UE drops all duplicate packets and the application receives only one copy from the OS.



(b) "Free-riding" retransmission attack: 1) The UE attaches a fake TCP header tunneling the real packet and sends it to a TCP proxy. 2) The core network recognizes the packet as retransmission and does not account for it. 3) The TCP tunneling proxy de-tunnels the packet and forwards it to the destination server. 4) The destination server accepts the packet thinking that it is communicating to the TCP proxy.

Figure 3: Attack scenarios that abuse cellular data accounting policies for TCP-level retransmission

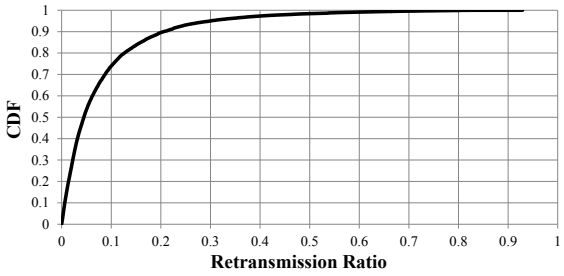


Figure 4: CDF of the retransmission ratios of the flows that experience any packet retransmission in a 3G network

retransmissions, an attacker can deplete a user's data plan, or if the provider ignores retransmissions, it is possible to tunnel traffic for free if the accounting system does not perform deep packet inspection (DPI). For the latter, we propose to hide our traffic inside of TCP retransmission packets. The basic idea is to send the traffic via a TCP proxy to the destination servers. A mobile client wraps the real TCP traffic in a fake TCP header that looks like a retransmission packet, and sends it to the TCP proxy, and the proxy de-tunnels the real TCP packet and forwards it to the destination. The traffic from the destination is again wrapped in a TCP header that uses an old sequence number by the proxy and is forwarded to the mobile client. This way, a real TCP session can be tunneled in a fake TCP session that avoids accounting.

3.2 Packet Retransmission in Cellular Networks

To determine the level of retransmission in real-world cellular networks, we measured the retransmission ratio at one of the largest cellular ISPs in South Korea. We mirrored the 3G traffic at one of 10 Gbps links just below a GGSN in Seoul, and inspected all TCP flows for 3 hours during the daily peak time (2012/09/29 9PM-0AM). We observed 134,574,018 flows with 6.64 TBs of IPv4 packets. Our monitoring system manages each TCP session by keeping track of TCP connection setup and teardown, sequence numbers, ACKs, and timeouts without a single packet drop during the measurement period.

Overall, we find that the retransmission ratio is reasonably low, which implies that the cellular networks are well-provisioned. Only 1.89% of the flows show a positive number of packet retransmissions during the period. This is in part because the majority of the flows are small (almost 90% of them are smaller than 32 KB) and are short-lived. However, we do find that some large flows experience severe packet retransmissions with as much as 93% of the packets in the flow being retransmitted as shown in Figure 4. This situation can be aggravated by poor provisioning, causing lost or delayed packets at the mobile station and forcing TCP retransmissions.

While our measurements imply that accounting for retransmissions would not incur a noticeable usage blowup for most subscribers for now, the users could be the victim of malicious retransmissions that inflate the usage. For example, attackers could participate in popular web sites as advertisers such that their advertisement content is served by a malicious server with a non-compliant TCP stack that intentionally retransmits TCP packets without waiting for timeouts. This way, the attacker can manipulate the accounting mechanism of competing ISPs or blow up the usage of a particular user. In our experiments in Section 4, we show that one can inflate the byte usage arbitrarily if the ISPs blindly account for retransmissions.

Cellular ISP	Test Client Device
AT&T (US)	Apple iPhone 4 (iOS 5.1.1 - 9B206)
Verizon (US)	Apple iPad 2 (iOS 5.1.1 - 9B206)
SKT (South Korea)	Galaxy S3 (Android 4.0.4)
KT (South Korea)	Galaxy S3 (Android 4.0.4)
LGU+ (South Korea)	Galaxy S3 (Android 4.0.4)

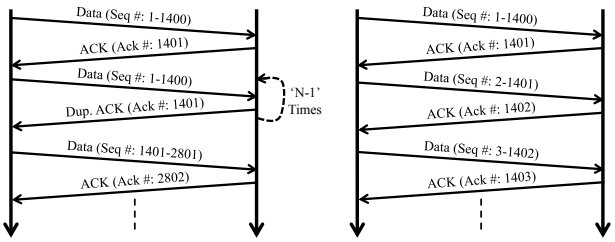
Table 1: Test client devices for each cellular ISP

4. RETRANSMISSION EXPERIMENTS

In this section, we run various tests to figure out the accounting policies currently being enforced in commercial cellular ISPs. Table 1 shows five large cellular ISPs in the U.S. and South Korea used in our tests as well as the test client devices and their OS versions. We download a file from our custom Web server that manipulates the TCP packets to test a number of retransmission scenarios, and verify whether the accounted volume by the ISP and the byte count in the captured packet trace at clients match.

4.1 Test Setup

To generate retransmission packets at will in the middle of a TCP connection, we build our own server that serves a web object. Our



(a) Controlled retransmissions (b) Quasi retransmissions

Figure 5: Example packet flows of our tests

custom web server accepts a regular TCP connection, processes a Web request, and serves the requested object. When a connection is established, instead of using the accepted TCP socket, the server opens a *raw* socket to read the IP packets from the client by filtering the port and the address, delivers the requested content, and sometimes injects retransmission packets to gauge the accounting policies. This way, we can create our own TCP/IP headers for each outgoing packet and confirm the ACK number from the client. For simplicity, our server maintains a TCP window size of one packet and does not implement congestion nor flow control.

In the client side, we use *wget* to fetch the content from our server. For accurate verification of the accounting volume, we either root or jailbreak our devices and run packet capture programs such as tcpdump [16] or pirni [17]. We collect all packet traces at clients during the test and compare the byte count with the accounted number provided by each ISP. After each download test, we turn off the cellular network interface on the client device and wait until the accounted data volume of the ISP is refreshed. We divide the measured volume into various categories such as normal ACKs without payload, normal data packets, duplicate ACKs, and retransmitted data packets. TCP packets for connection handshake and teardown, and other background traffic are carefully excluded from the results by subtracting them from the total value.

4.2 Experiments and Results

We use five main experiments to determine the accounting policies of various cellular service providers regarding DNS packets and TCP retransmission packets. We include DNS tests to verify the accounting policy loophole reported by recent works [9,10] and to reflect the policy into the measured results. Each test is run three times and we show the average value. The ISPs are addressed by number, with ISP-1, 2 and 3 based in South Korea and ISP-4 and ISP-5 being based in the United States. We note that ISP-1 and ISP-4 provide an accounting granularity of 1 KB, ISP-2 and ISP-3, a granularity of 100 KB, and ISP-5, a granularity of 1 MB.

4.2.1 DNS Packet Accounting

Peng et. al. recently report that packets with the DNS port are considered as a free service and are not accounted for in a number of ISPs [9, 10]. Our first step is to verify this claim by running DNS lookups of 10,000 different domain names and comparing the data volume seen by the client and by the ISP. In our measurements in October 2012, we found that ISPs 1, 2 and 3 do not account for UDP-based DNS packets, but we were surprised to discover that ISP-4 and 5 account for all DNS packets, suggesting that some providers have already started to react to the DNS tunneling reports. In addition, we check whether the TCP packets going through port 53 (DNS) are considered free by downloading some content on the DNS port. We confirm that ISPs 1, 2, 3 that do not account for

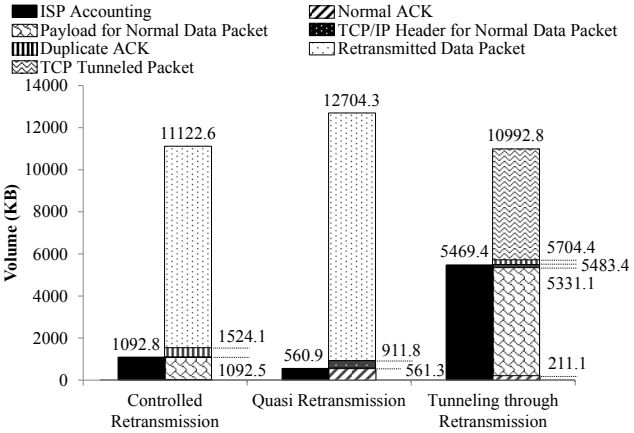


Figure 6: Experiment results of ISP-1

UDP-based DNS packets do charge for all TCP packets on port 53, thus DNS tunneling attacks are not possible with these ISPs.

4.2.2 Content Transfer without Packet Loss

As a base case, we compare the measurement results by downloading a file over a reliable link without any intentional packet retransmissions. This test is to verify whether the ISPs account for the traffic accurately in a normal situation with little packet loss. We calculate the theoretical value and compare it with the ISP's accounted volume as explained below. For each test, we confirm the absence of packet retransmissions by checking the captured packet traces. We compare the accounting values from three sources; the ISP, the mobile client, and the theoretical model. We calculate the theoretical value by taking into account the TCP connection setup/teardown, ACK and data packets including headers from layer 3 and up, and the background traffic from other local processes running on the mobile client. We find that all ISPs account for the proper amount of the data volume in this test, confirming the accurate accounting in the base case.

4.2.3 Controlled Retransmissions

We also run the test that intentionally injects retransmission packets between each data packet. We initiate a TCP connection from the mobile client, make sure that the server goes through the TCP-handshake, and then have the server send a pre-determined number of retransmission packets per each data packet. From the size of the original data to be transmitted, we can easily calculate the total volume, as the retransmissions will act as a simple multiplier. Figure 5(a) shows the test scenario.

We wait for the mobile device to ACK a retransmitted packet before sending another one to ensure proper reception. We test each ISP with 9 retransmissions per each data packet (e.g., 10 identical data packets in total, a blowup by a factor of 10 in the real payload).

We discover that only two ISPs in South Korea (ISP-1 and ISP-2) do not account for the retransmission packets while the others do. The two leftmost bars in Figure 6 and Figure 7 show the results of ISP-1 and ISP-2. We download a 1 MB file for ISP-1 while we use a 10 MB file for ISP-2 since ISP-2 supports 100 KB accounting granularity. Interestingly, we see that the accounting policy differs from ISP to ISP even in the same country. ISP-3 in South Korea accounts for every packet regardless of retransmission. We also note that the accounting policies for ISP-1 and ISP-2 are slightly different. While they both ignore retransmitted data packets, ISP-2 accounts for duplicate ACKs while ISP-1 ignore them for account-

ing. We confirm that the other three ISPs count every retransmission, showing a blowup by a factor of 10 from the original file size in the accounted volume. For this reason, we leave out the graphs for these ISPs. This test implies that the users in these ISPs can be the victim of usage-inflation attack.

4.2.4 *Quasi Retransmissions*

The next question is how the service providers would account for partial retransmissions where the next packet overlaps partially with the previous packet. More specifically, the server increments the current window by just one byte, but the data content is much larger. The resulting stream is one where the sequence numbers are not directly repeated, but the data content largely overlaps. This will tell us if the service provider accounts by data packets, or takes the actual data window of a TCP packet into account. We send a small amount of application layer data (10KB, 75KB), but make sure that the packet window is only incremented by one byte, although the payload of each packet contains over 1.3 KB of content. We omit the ISPs that charge for retransmissions since they only account for the complete volume anyway.

The two middle bars in Figure 6 show the result for ISP-1. We see that the accounted value is actually less than the data volume excluding the retransmitted data packets. This is due to the ISP not charging the TCP/IP headers for data with partially-retransmitted payload. ACK packets are all counted since each ACK packet has its acknowledgement number increased by one. On the contrary, ISP-2 (middle bars in Figure 7) accounts for all TCP/IP headers but not the retransmitted payload itself. This could be explained by an ISP that checks the sequence number and the packet length to identify the actual data volume but charges for the entire header since there is at least one byte of new payload.

4.2.5 *Tunneling through Retransmissions*

Finally, we measure if the service providers verify that the data content of retransmissions do in fact contain a copy of the previous packet's payload data. If they only rely on the TCP headers, an attacker could set up a covert channel in the payload field of the TCP retransmission packets to avoid data charges. We were also careful to set the sequence number of the retransmission packets to be within the range of the most recently-ACK'ed packet to prevent middleboxes or the recipient's OS kernel from dropping packets with old sequence numbers.

The two rightmost bars in Figures 6 and 7 show that both ISP-1 and ISP-2 do not account for retransmitted packets with different payload. This makes intuitive sense since deep inspection of the TCP payload of every packet would be space and time consuming. From this test, we conclude that all ISPs that do not account for retransmitted packets are open to TCP-retransmission tunneling.

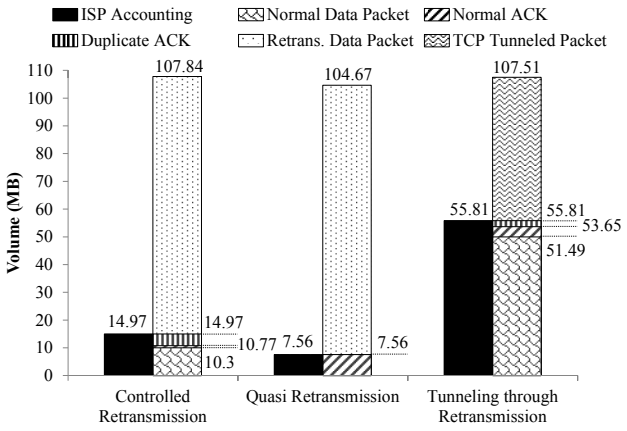


Figure 7: Experiment results of ISP-2

5. MITIGATION

To provide fair accounting, one can decide to account for retransmitted packets but block the "usage-inflation" attack or decide not to account for retransmitted packets but defend against the "free-riding" attack. The former makes sense if we can assume a low legitimate packet loss rate throughout the cellular infrastructure, but it could penalize users that are already getting poor coverage service. Instead, we focus on the latter here and briefly propose three plausible mitigation techniques against "free-riders".

Detection of Abnormal Retransmission. The cellular ISP may set a limit on the number or ratio of retransmission packets per flow. The GSN detects an abnormal flow with the number of retransmissions exceeding a certain threshold, and alerts the ISP of a possible attack. Once a flow turns out to abuse the retransmission policy, the ISP can decide to either charge all retransmission packets or explicitly close the connection. This approach is attractive since it requires only small states per each flow (e.g., number of retransmissions per packet, retransmission ratio, etc.), not causing much overhead on the CN. However, the major disadvantage of this method is that it could incur a false-positive alarm. We have shown that even a legitimate flow experiences the retransmission ratio of 93% in poor cellular network environments. Therefore, naively setting a threshold could result in penalizing innocent users or tunneling attacks, depending on the value of the threshold.

Deterministic DPI. A more accurate solution is to run DPI on all TCP flows where the system temporarily stores the content in every flow and compares the payload if it detects a retransmission. A bytewise comparison over the retransmitted range can ensure identical retransmission content. This method is advantageous in that it can completely remove the false-positive alarm. One obvious drawback, however, is that it could incur high system overheads of managing the buffer of every TCP flow. In our measurement at a 10 Gbps 3G backbone link, we see up to 1.3 million new TCP flows per minute with 270 K concurrent flows at peak. In the worst case, an accounting system should handle tens of millions of packets per second per 10 Gbps link. We are currently building a middlebox system that can manage 100Ks of concurrent flows for a 10 Gbps link by careful buffer memory management and parallel processing on a multicore system. However, it would be still challenging or costly if it requires multiple load-balanced machines.

As a hybrid solution, one might set a small threshold for detecting abnormal retransmissions and run deterministic DPI only if the retransmission ratio is beyond the threshold. This would greatly reduce the system overhead by bypassing the majority of normal flows, and could detect long-lived flows that tunnel packets. However, it is still not perfect if a sophisticated attacker carefully manages small flows that do not trigger the alarm.

Probabilistic DPI. To reduce the memory requirements, we propose a lightweight method where the CN only has to inspect a random part of the TCP payload. Thus, between two candidate retransmissions, this method needs to (1) look in the same places in the payload and (2) find the same bytes at those positions. For step (1), we can use the sequence number as an index into a table containing n random locations per packet where bytes will be extracted from the payload. We could use a random number generator with a secret number as a seed to determine the n-byte locations per each flow. For step (2), once we have extracted some bytes from the same location on both packets, we can compute the difference be-

tween those n-byte sequences. If it is anything other than 0, we can confirm that the retransmission payload is different from the original payload.

We note that we have only reduced the space complexity by a constant factor, from a full TCP payload to an n-byte representation, but storing a fraction of the payloads at minimal computing costs will help in the real-world implementation. The probability of collisions between the original payload and an arbitrary payload decreases exponentially as n increases, making the scheme quite space efficient. We also note that it would increase false negatives if n is too small. We plan to identify the appropriate n by analyzing multiple variables in TCP flows, including the average payload length, the probability of overlapping sequence number ranges in the retransmitted packets, the number of concurrent flows, and the average congestion window size.

6. RELATED WORKS

Peng et. al. have recently reported loopholes in some cellular ISPs that allow attackers to obtain free cellular Internet access by tunneling the data on the DNS port, since DNS is viewed as a free infrastructure service and the payloads are not inspected [9, 10]. In our measurements, we find that TCP packets on the DNS port get charged even for the ISPs that allow free UDP-based DNS packets. Running DPI on the DNS packet would incur relatively small overheads since the number of DNS packets is typically much smaller that that of other data packets and each DNS packet is just a few hundred bytes. In addition, we have shown that there is a more fundamental issue in cellular data accounting for TCP-level packet retransmission. Building a DPI-based cellular accounting system that analyzes every TCP packet going through the CN remains to be a challenge.

Lee provides one of the early works that measure the retransmission rate over CDMA 1x EV-DO service [18]. The author finds that the average retransmission rate of a flow reaches up to 4.7% with 92% burst retransmissions in the uplink. The retransmission ratio shows a similar characteristic to our experiment where larger flows are more likely to be affected by the retransmissions. Won et. al. show that in CDMA networks, almost 80% of the total packets captured in the link are retransmission packets [19]. They also find that 38% of the TCP sessions have 9 out of 10 packets as retransmission packets, which implies that our attack could be effective in the CDMA network as well. Jang et. al. look at the retransmissions in HSDPA networks (3G, 3.5G) from moving cars and express trains [20]. Their results show that when the vehicles are on the move, the retransmission ratio increases up to 71 times higher than that in the stationary case, implying that the users with higher mobility will have to pay more if retransmission packets are accounted. Gember et. al. measure the retransmission rate in Wi-Fi networks at a university campus [21]. They show that even in the less-congested wireless network, 5% of flows have one or more retransmission packets where 80% of them are due to packet loss.

7. CONCLUSION

We have shown that due to the current design of the cellular data architecture and transport layer reliability mechanisms using retransmissions, the accounting policies either leave the user vulnerable to data depletion attacks, or cause the ISP to be vulnerable to service charge evasion due to tunneling through retransmissions. We have measured the effect of retransmissions on five major ISPs in two countries, demonstrating the possibility of data depletion attacks, or free-riding tunneling. We believe that it is possible for ISPs to provide a fair accounting of traffic usage while preventing free-of-charge abuse, and have proposed possible mitigations that could be implemented with relatively low costs.

8. ACKNOWLEDGMENTS

We appreciate insightful comments from anonymous reviewers and thank our shepherd, Ramon Caceres, for his help on improving the paper. This research was supported by the KCC (Korea Communications Commission), Korea, under the R&D program supervised by the KCA (Korea Communications Agency), KCA-2013-12911-05003.

9. REFERENCES

[1] CISCO. Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2011-2016. Technical report, 2012.

[2] ITU. ICT Facts and Figures. Technical report, 2011.

[3] Ericsson. Traffic and Market Report. Technical report, 2012.

[4] M. Jurvansuu, J. Prokkola, M. Hanski, and P. Perälä. HSDPA Performance in Live Networks. In *Proceedings of IEEE International Conference on Communications*, 2007.

[5] G. Maier, F. Schneider, and A. Feldmann. A First Look at Mobile Hand-Held Device Traffic. In *Proceedings of the Passive and Active Measurement*, 2010.

[6] 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 *Proceedings of the International Conference on Mobile Systems, Applications, and Services*, 2012.

[7] J. Erman, A. Gerber, K. K. Ramakrishnan, S. Sen, and O. Spatscheck. Over The Top Video: The Gorilla in Cellular Networks. In *Proceedings of the ACM SIGCOMM Conference on Internet Measurement Conference*, 2011.

[8] J. Postel. Transmission Control Protocol. RFC 793 (Standard), September 1981. Updated by RFCs 1122, 3168, 6093, 6528.

[9] C. Peng, G. Tu, C. Li, and S. Lu. Can We Pay for What We Get in 3G Data Access? In *Proceedings of Annual International Conference on Mobile Computing and Networking*, 2012.

[10] C. Peng, C. Li, G. Tu, S. Lu, and L. Zhang. Mobile Data Charging: New Attacks and Countermeasures. In *Proceedings of ACM Conference on Computer and Communications Security*, 2012.

[11] 3GPP. UMTS. `http://www.3gpp.org/Technologies/Keywords-Acronyms/article/umts`.

[12] 3GPP. LTE. `http://www.3gpp.org/LTE/`.

[13] 3GPP. ETSI TS 132 200. Telecommmunication management; Charging management; Charging principles.

[14] 3GPP. ETSI TS 132 215. Telecommunication management; Charging management; Charging data description for the PS domain.

[15] 3GPP. ETSI TS 129 060. General Packet Radio Service; GPRS Tunnelling Protocol across the Gn and Gp interface.

[16] Gadgetcat. tcpdump on Android, 2011. `http://gadgetcat.wordpress.com/2011/09/11/tcpdump-on-android/`.

[17] n1mda-dev. Pirni native iPhone ARP spoofer and network sniffer. `http://code.google.com/p/n1mda-dev/wiki/PirniUsageGuide`.

[18] Y. Lee. Measured TCP Performance in CDMA 1x EV-DO Network. In *Proceedings of the Passive and Active Measurement*, 2005.

[19] Y. J. Won, B. Park, S. Hong, K. Jung, H. Ju, and J. Hong. Measurement Analysis of Mobile Data Networks. In *Proceedings of the Passive and Active Measurement*, 2007.

[20] K. Jang, M. Han, S. Cho, H. Ryu, J. Lee, Y. Lee, and S. Moon. 3G and 3.5G Wireless Network Performance Measured from Moving Cars and High-Speed Trains. In *Proceedings of ACM Workshop on Mobile Internet Through Cellular Networks: Operations, Challenges and Solutions*, 2009.

[21] A. Gember, A. Anand, and A. Akella. A Comparative Study of Handheld and Non-Handheld Traffic in Campus Wi-Fi Networks. In *Proceedings of the Passive and Active Measurement*, 2011.

How is Energy Consumed in Smartphone Display Applications?

Xiang Chen, Yiran Chen
ECE Department, University of Pittsburgh
Pittsburgh, PA, USA 15261
+1 (412) 624-5836
{xic33, yic52}@pitt.edu

Zhan Ma, Felix C. A. Fernandes
Samsung Telecommunication America
Richardson, TX, USA 75082
+1(972)761-7134
{zhan.ma, ffernandes}@sta.samsung.com

ABSTRACT
Smartphones have emerged as a popular and frequently used platform for the consumption of multimedia. New display technologies, such as AMOLED, have been recently introduced to smartphones to fulfill the requirements of these multimedia applications. However, as an AMOLED screen's power consumption is determined by the display content, such applications are often limited by the battery life of the device they are running on, inspiring many researches to develop new power management schemes. In this work, we evaluate the power consumption of several applications on a series of Samsung smartphones and take a deep look into AMOLED's power consumption and its relative contributions for multimedia apps. We improve AMOLED power analysis by considering the dynamic factors in displaying, and analyze the individual factors affecting power consumption when streaming video, playing a video game, and recording video via a device's built-in camera. Our detailed measurements refine the power analysis of smartphones and reveal some interesting perspectives regarding the power consumption of AMOLED displays in multimedia applications.

Categories and Subject Descriptors
K.6.2 [**Management OF Computing and Information Systems**]: Installation Management – *Performance and usage measurement.*

General Terms
Algorithms, Management, Measurement, Performance, Design.

Keywords
OLED display, Smartphone, AMOLED, Video power.

1. INTRODUCTION
Smartphones now play a large role in almost every aspect of our daily lives, being utilized in activities such as communication, personal planning, and entertainment. Although the complexity and capabilities of these devices continues to grow at an amazing pace, smartphones are now expected to continually become lighter and slimmer. When combined with power-hungry multimedia applications, the limited battery capacity allowed by these expectations now motivates significant investment into smartphone power management research.

Multimedia applications, such as streaming video players, games, and image and video capturing tools, now comprise a considerable portion of the daily usage of smartphones. The power consumption of these applications depends heavily on the type of display technology being used, which also plays an important role in human-machine interaction. Many researches have already studied the power and performance of different display technologies [1].

Very recently, AMOLED (Active-Matrix Organic Light Emitting Diode) display panels have begun to replace conventional LCD (Liquid Crystal Display) technology in mainstream smartphones. Compared to LCD, AMOLED offers much better display quality and higher power efficiency because of its unique lighting mechanism. This unique property has led to much research involving the power evaluation and modeling of AMOLED as well as the performance variance among different AMOLED panel designs. However, due to the large variety of AMOLED panel designs and the fast pace of smartphone software development, most of this research only aims at a particular smartphone device or application. It is not clear whether there is any significant power efficiency improvement between different generations of AMOLED display technology or if it is possible to obtain an overview display power efficiency under different applications.

In this work, we evaluate the power consumption of the displays of several high-end Samsung smartphone models. In addition to this, we discuss the power modeling of different generations of AMOLED screens and conduct a detailed power analysis of several popular multimedia applications. Based on the data that was collected, we draw some interesting conclusions which may influence future research in the field. For example:

- Subsequent generations of AMOLED products do not yield the significant per-unit improvement in power efficiency that was expected. The power difference between AMOLED products is mainly realized by design metrics like size and sub-pixel.
- The power consumption of chromatic color can be efficiently reduced by implementing a dynamic color tuning technique.
- The power consumed during the video decoding process is responsible for only a small portion of the total power required for the end user to actually view the video.
- In video games, the AMOLED display's power consumption varies greatly, and it is relatively small in some cases when compared to the overall system power consumption.
- The power consumed by the AMOLED during video recording is relatively small compared to the overall power consumption.

The remainder of our paper is organized as follows: Section 2 presents previous related work; Section 3 gives our adopted methodology for power evaluation, including the devices under test (DUTs) and the test environment setup; Section 4 models the power consumptions of several different AMOLED panels on Samsung smartphones; Section 5 presents the power analysis' of video streaming players, video games, and camera recording, respectively; Section 6 concludes our work.

ACM HotMobile'13, February 26-27, 2013, Jekyll Island, Georgia, USA.

2. RELATED WORK

Since its development, AMOLED technology has been the topic of a great deal of research, much of which has attempted to describe and model its power characteristics. The authors of [2] presented a classic AMOLED power modeling in smartphone, which is adopted in this paper. The authors of [3] evaluated the AMOLED power model's accuracy in practical performance and helped application developers to improve the energy efficiency of their smartphone applications. OLED power model is part of the system power model. The display modeling of AMOLED has also been integrated into a system level power monitor and analyzer, as done in [8] and [9].

Power optimization schemes based on AMOLED displays have also been studied from the applications point of view. The authors of [15] reduced the power consumed while displaying an arbitrary image through the use of OLED's dynamic voltage scaling (DVS) technology, while the authors of [4] extended this DVS scheme into video streams. The authors of [7] extended the dynamic gamma correction power saving scheme for video games to the AMOLED platform. The author of [5] shows OLED screens contribute significantly to the energy consumption of web browser and then applies the optimization techniques from [6] to reduce it. These works pointed out that, while a smartphone spends most time idle when web browsing, streaming video, playing video game, and camera recording, the smartphone demands significant system resources in a continuous manner. However, we find the power contribution made by the OLED screen can be small.

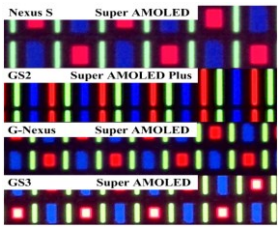
3. METHODOLOGY

3.1 Devices Under Test (DUTs)

We examined five Samsung products, including the Nexus S (released in 2010), the Galaxy S1, Nexus, and S2 (released in 2011), and the Galaxy S3 (released in 2012). During the tests, the devices ran the Android operating system and a suite of test applications. The screens of all the tested devices are all based on AMOLED technology, though they are built with different display sizes, resolutions, and technology generations, i.e., Super AMOLED, Super AMOLED Plus and Super AMOLED HD, respectively. These screens well represent the evolution of Samsung's OLED technology in recent years.

3.2 Power Evaluation Setup

Many smartphone power models have been proposed in recent research and are included in the embedded Android power monitor APIs [8][9][10]. However, these models generally have limited adaptability to the many variations of available hardware configurations, incurring inevitable run-time evaluation errors [11]. Rather than calculating power information from software, we utilized an external power monitor to directly measure the battery behavior for power consumption breakdown with a series of contrast experiments.



(a)

Smartphone	Panel Size	Resolution
Nexus S	4.0"	480*800
Galaxy S2	4.5"	480*800
Galaxy Nexus	4.65"	720*1280
Galaxy S3	4.8"	720*1280

(b)

Fig. 1. (a) AMOLED sub-pixel matrix; (b) Smartphone AMOLED screen specs.

We adopted a mobile device power monitor produced by Monsoon Inc. for real time power measurement. It supports a sampling frequency up to 5 kHz and can be used to replace the battery as the power supply to the smartphone. The power monitor is directly connected to the smartphone and records a power histogram. During the power evaluation of the display modules, we disabled all unnecessary system services that may have caused any significant power consumption noise, including 4G and Wi-Fi network communication, background services, and power optimization applications. The contrast experiments are designed to separate the power consumption of display from that of the system, e.g., CPU and GPU. Most power consumption test are completed while the device's screen brightness is set to maximum. The specific test bench details will be presented in later sections.

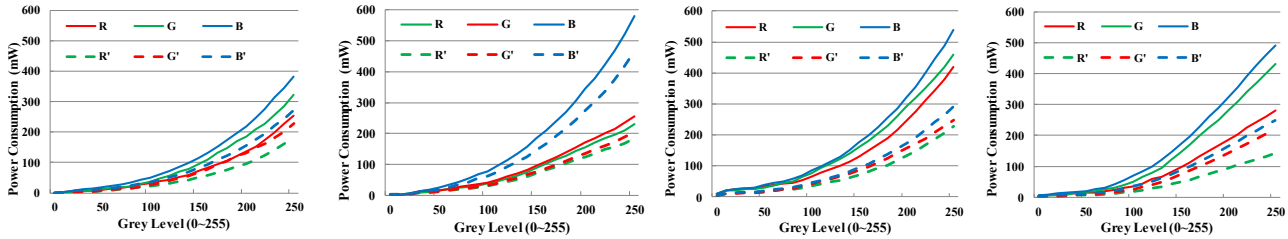
4. AMOLED DISPLAY EXPLORATION

4.1 AMOLED Power Evaluation on Sub-pixel

Different from previous research, instead of directly measuring and then comparing the power consumption of each entire screen, we attempt to normalize the displays and compare the per-unit power efficiency for different AMOLED products. Because of this we found that, in contrast to the conclusions drawn by previous research, when comparing the power efficiency between different generations of AMOLED technology:

Subsequent generations of AMOLED products do not yield the significant per-unit improvement in power efficiency that was expected. The power difference between AMOLED products is mainly realized by design metrics like size and sub-pixel matrix.

A smartphone display is composed of many individual pixels. The color of a pixel is constructed via the combination of a few basic colors (e.g., red, green, blue, or RGB), which themselves are the only colors directly emitted from the display unit via tiny sub-pixels. The display unit is defined by the arrangement of these sub-pixels, which is known as the sub-pixel matrix design. As shown in Fig. 1(a), in the AMOLED displays we tested, every pixel has three sub-pixels corresponding to the basic colors of RGB color space. Unlike LCD technology, the power consumption of an AMOLED



(a) (b) (c) (d)

Fig. 2. AMOLED screen power consumption: (a) Nexus S; (b) Galaxy S2; (c) Galaxy Nexus; (d) Galaxy S3
Solid lines are the original overall screen power consumption, dash lines are the screen power consumption normalized to 4 inch^2.

screen is highly color dependent. The power consumption of each pixel varies with the chromatic color composition that is being displayed, as not only do more intense colors require more power to display, but each individual sub-pixel has its own unique power distribution curve. Hence, various sub-pixel matrix designs are proposed to balance the lighting efficiency and display quality by adopting different alignments, area-to-area ratios, and etc. of individual sub-pixels. Fig. 1(a) illustrates the contrasting sub-pixel matrix designs of Super AMOELD Plus (based on identical RGB strips) and Super AMOLED (based on PenTile design) [12].

In many prior works involving AMOLED screens, the pixel-level power modeling is often expressed as: $P_{pixel} = f(R)+h(G)+g(B)$. This models the power consumption of a pixel as the summation of the power consumption of the individual RGB sub-pixels [2]. $f()$, $h()$, and $g()$ are the color component dependent functions and R, G, and B are the input signals, which are usually represented as a grey level of [0, 255]. In Samsung OLED products, a standard gamma correction of 2.2 is applied to the input signals in order to improve the display quality. Thus, the relationship between the power consumption of the AMOLED screen and the grey level becomes nonlinear, as shown in Fig. 2.

Fig. 2 also shows that the power models of different display modules vary significantly. One example of this is the difference between the Super AMOLED or PenTile AMOLED, used in Galaxy S3 and the Super AMOLED Plus used in Galaxy S2: the power consumption of the PenTile design in Super AMOLED is much more balanced among the different colors than the traditional RGB strips utilized in Super AMOLED Plus. In addition to this, display panels with the PenTile design also show an averagely lower power efficiency than traditional RGB strip. However, as PenTile design aligns more low power consuming green sub-pixels in the pixel matrix, it's supposed to be more power efficient. Hence, we have to also take the display panel size into consideration to compare the power efficiency.

In many existing AMOLED power measurements and models, the power consumption of a display panel is evaluated as a whole. The many different panel sizes and sub-pixel alignments available today make it very difficult to conduct a fair comparison of the power efficiency between different AMOLED technology generations. Therefore, we focused on analyzing the sub-pixel power efficiency.

To accomplish this task, we first normalized the area of each individual sub-pixel to account for the size difference between sub-pixels in PenTile technology. The power consumption of the entire AMOLED screen while displaying each basic color at every grey level was then recorded and scaled to represent a screen size of 4 inch^2. The normalized power consumption of the sub-pixels in each scaled display is depicted as the dashed lines in Fig. 2.

Although some prior work has claimed that the power efficiency of AMOLED screens has as much as doubled between generations, i.e., from the Galaxy S2 to the Galaxy S3 [18], we found that the practical power improvement is not nearly that significant. For example, when comparing to the oldest AMOLED screen (in the Nexus S) to the latest (in the Galaxy S3), a jump of two generations, the highest pixel-level power efficiency is between only 5.7% and 29.5% improved at a grey level of 50. Interestingly, we found that the most significant factor that contributes to the power consumption difference between the different smartphone models is the sub-pixel area ratio. For example, although all the Super AMOLED display panels follow G-R-G-B sub-pixel matrix design, the sub-pixel area ratio is slightly adjusted in the different models to accommodate the specific screen size and pixel density, leading to different power models.

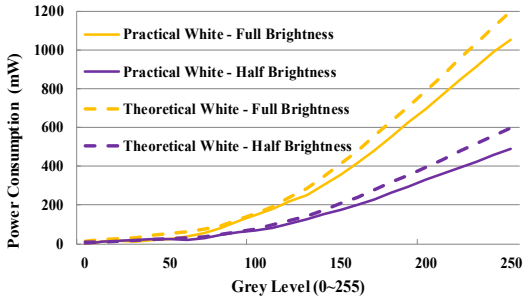


Fig. 3. System chromatic color tone remapping.

4.2 Dynamic Color Tuning

Based on the color power measurement, we also found that:

In the operation of an AMOLED screen on a smartphone, the actual displayed color and intensity may not be representative of the original raw RGB composition being sent to the screen as it may have been modified by a dynamic color tuning system in order to minimize power consumption.

The previous single color evaluation is realized by adjusting the grey level of individual RGB color component while disabling the other display channels. However, when we measured the chromatic color's power consumption in Galaxy S3, we found that it is not simply the summation of the individual RGB channels' pixel power model, which was discussed in Section 3.1.

We examined the power consumption of the AMOLED screen when its color changes from black to white. At every measured color composition, the same grey levels are applied to each sub-pixel. As shown in Fig. 3, at a grey level of 100 and above, the theoretical power consumption that is calculated by the existing simple power model is 7% to 14% higher than the practical (measured) power consumption under full brightness and 9% to 22% higher under half brightness. Similar phenomena were observed when displaying arbitrary chromatic colors, such as aqua, pink, purple, and etc.

In practice, it is very difficult to measure the power consumption of an AMOLED screen in the 24-bit RGB color space and derive a generic power model. However, it is still possible to integrate a simplified model into the smartphone for AMOLED screen power optimization. In fact, in Samsung's display system, an image processing engine called MDNIe is implemented by a designated chipset just in front of the graphic buffer to the display panel [13]. One application example of MDNIe is dynamic color tuning: the color composition being displayed on the screen can be adjusted by MDNIe to decrease power consumption while maintaining contrast levels. Hence, this system integrated dynamic color tuning has achieved power savings by color tuning effectively.

5. POWER ANALYSIS ON DISPLAY RELATED SMARTPHONE APPLICATION

5.1 Video Power Performance

Although OLED technology has substantially improved the power efficiency of displays when compared to traditional LCD screens [14], the display panel is still one of the most power-consuming components in a smartphone [15]. Applications with dynamic display contents, e.g., streaming video player, are considered as being energy-hungry by previous research.

However, in this work, we found that:

In video stream player, the AMOLED screen's power consumption is highly content dependent. The video decoding process contributes very marginal power consumption to the application.

We examined a set of pure video streams without audio track and analyzed the composition of the application's power consumption. We also conducted a breakdown of the screen's power consumption over each functional component. To better represent typical display content differences, we followed YouTube's video category and selected four types of video streams: music videos, sports, gameplay videos, and news reports. All of the video clips tested are selected and downloaded from YouTube. Most of these video streams are between 2-3 minutes in length and vary between 1000 and 2500 frames.

5.1.1 Power consumption with different display contents

Because of the previously discussed color-dependent power model associated with AMOLED pixels, the display content directly determines the power consumption of an AMOLED screen. We evaluated 50 local video streams in each category on a Samsung Galaxy S3 in the field study. All of the video streams were encoded with a bit rate of 0.8 Mbps without a sound track. The refresh frame rates and display areas were configured differently to evaluate the system decoding performance. The overall power consumption of the device was recorded while at the same time monitoring the individual component power breakdown.

The average power consumptions of each category of video are shown in Table 1. In full screen tests, videos are stretched to fill the entire screen, while in original size tests the videos are displayed on the screen in a letterbox with a resolution of 640x360 (the rest of the space on screen is black). The original size test is used to simulate the small display windows embedded in other application's UI's, e.g., Facebook and YouTube.

Table 2 gives the contrast test results, which are measured by disabling the display output and allowing it to only display a black screen. In these tests, the power consumption of the device is only influenced by the video processing and other background operations. As expected, the contrast values are not dependent on the display content and are almost identical for different content samples with the same decoding configuration. By subtracting the power consumption in Table 2 from corresponding items in Table 1, we can derive the power consumed by the display panel itself.

Our results show that the most power consuming video content category is Sports. At 60fps and 30fps, the AMOLED screen consumes 29% and 32% of total smartphone power, respectively. Such high power consumption comes from the bright color tone and complex textures, which require high luminance and balanced color

Table 1. Video stream power consumption (mW)

Video	1280x720 (full screen)		640x360 (full screen)		640x360 (original size)	
	30fps	60fps	30fps	60fps	30fps	60fps
Music	786.4	975.8	682.9	783.7	594.9	698.1
Sports	960.4	1172.1	855.3	953.2	652.4	751.9
Game	869.0	1061.2	786.5	889.1	628.2	730.8
News	901.3	1124.3	802.7	915.7	612.3	725.5

Table 2. Contrast power consumption (mW)

Video	1280x720 (full screen)		640x360 (full screen)		640x360 (original size)	
	30fps	60fps	30fps	60fps	30fps	60fps
Music	646.9	820.8	543.8	648.2	544.8	649.4
Sports	646.8	823.2	542.5	644.3	548.7	649.0
Game	644.3	820.6	543.5	643.7	548.0	648.8
News	647.5	822.1	545.2	644.2	548.2	649.0

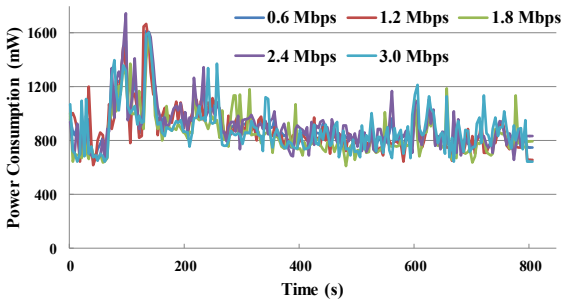


Fig. 4 System chromatic color tone remapping

tuning. As a comparison, the least power consuming video content category is music video, which consumes only 15% and 17% of total smartphone power at 60fps and 30fps, respectively.

Our results also show that the power consumption of the AMOLED screen decreases considerably when the resolution of the displayed video stream degrades: the ratio of the power consumed by the AMOLED screen to that of the entire smartphone decreases to between 7% and 15%. This indicates that while streaming an online video, AMOLED screens consume quite a small portion of the overall smartphone power, especially when compared to the potential power cost of network, which can require up to 2W.

Moreover, we simulated approximately 200 video streams under each category to obtain the power consumption statistics of AMOLED screens. Our simulated results show that power consumption of the AMOLED screen is between 74.6mW and 374.2mW during full screen viewing and 37.2mW to 90.7mW during original size viewing. When compared to the contrast power consumption shown in Table 2, we propose that:

The power consumption of an AMOLED screen is highly content dependent and may have relatively little impact on the overall power consumption of video streaming applications.

5.1.2 Video stream decoding cost

We also analyzed the power consumption of the video decoding process. The impact of display resolution, frame rate, and encoding bit rate are included in our analysis.

Similar to the experiments presented in Section 4.1.1, two display resolutions of 1280x720 (full screen) and 640x360 are adopted in our tests. For the same display content, bit rate, and fps, the observed video decoding power consumption is almost identical when the stream is stretched to full screen from 640x360 and when it is viewed at original, letterboxed size. This means that the full screen pixel stretching in the display buffer consumes very little power for low resolutions. However, a large power rise (~ 100mW) is observed if the resolution increases from 640x360 to 1280x720, as shown in Table 2. This is due to the significant power required during the decoding process for high-resolution video.

The largest power consumption difference occurs when the frame rate changes. The increase in the system power consumption at a fast frame rate comes from extra workload placed on the CPU. For example, the power consumption of the decoding process at 60 fps and a resolution of 640x360 is about 100mW higher than that of one at 30fps. When the resolution rises to 1280x720, the difference becomes even more pronounced at 200mW.

We also tested 50 complex CG videos with the same content, time length, frame rate and resolution but different display quality, i.e., the bit rate. In our tested videos, the bit rate varies from 0.6 Mbps to 3.0Mbps. Although the videos share the same resolution, the

low-quality ones include many mosaics and the corresponding video compression can be easier, which led to the lower bit rate. However, we found that the low bit rate and aggressive compression do not introduce any significant changes in power consumption: as is shown in the power track example in Fig. 4, there are almost no power differences between the different bit rates. Combined with our previous tests, this led us to the conclusion that higher display resolutions incur a much more significant increase in power consumption than an increased bitrate. Therefore, we propose that:

Fast frame rate and high resolution introduce significant video decoding power consumption. However, when compared to the whole system, this power portion is not significant.

5.2 Game Power Performance

Due to the large volume of graphic computation on CPU/GPU and high display quality requirement, video games have become one of the most power-consuming application types in smartphones. We also measured the game power performance, based on the above observation, we propose that:

Smartphone games consume a large amount of power. This power is mainly consumed by background computation while significant extra power may be required by user interaction. However, the AMOLED power consumption is generally small in comparison.

We first measured the power consumptions of 20 of the most popular games from the Google Play store on the Galaxy S3 platform. To enable easy interaction, most of the games are composed of big objectives with bright colors. This generates an increase in the power consumption of the AMOLED display. Our measurements show that the power consumption of the whole smartphone is generally within the range of between 1140mW and 1750mW when user interaction is disabled.

Some video games require user interaction that involves frequent sensor operations (e.g., tilt to move or touchscreen input). For instance, the power consumption of the device while playing Angry Birds and Fruit Ninja was increased to 1640mW and 2220mW, respectively, during normal operations. Compared to other applications like web browsing (with a power consumption in the range of 600mW to 1500mW) and video players, these video games require large amounts of power. The display power required with each of another 200 Android video games was simulated with the Galaxy S3 AMOLED power model. The simulations were made with official game trailer videos from YouTube. The average power consumption of the AMOLED was between 50.3mW~446.4mW.

Finally, we analyzed power utilized by background computation while playing the open-source video game Quake 3 on the Android platform. Although it is an old game, it still offers reasonable game complexity and display quality on present smartphone platform. By modifying the code, we were able to obtain the power breakdown

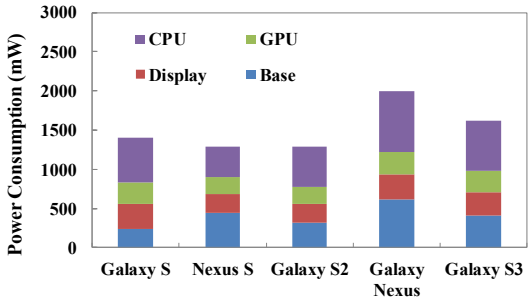


Fig. 5 Power component breakdown example of video game.

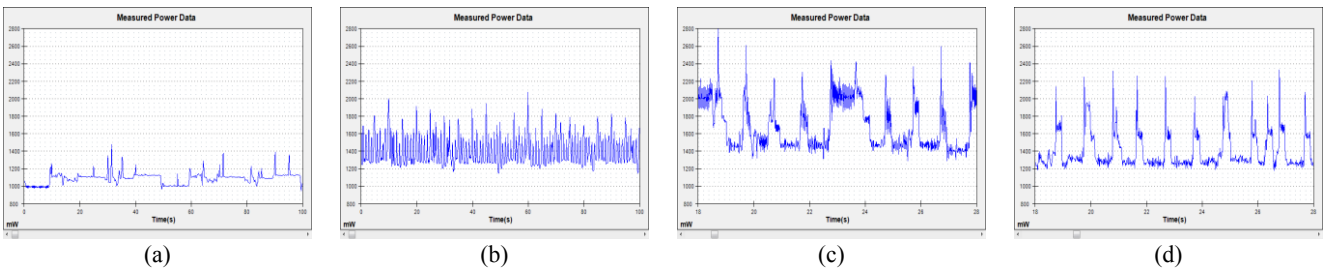
of the game over different hardware models. We chose five different smartphone models to cover the existing GPU/CPU configurations and display panel designs, as shown in Fig. 5. Although some models share the same chipset, e.g., the popular GPUs – PowerVR and Adreno (e.g., Galaxy Nexus uses PowerVR and ARM9; Galaxy S2 and S3 use Adreno and the CPU from Qualcomm), a variance in power performance is still observed. For example, the power supporting necessary background system services varies between 1245mW and 2140mW, which is much higher than the normal power level of other applications, such as streaming video players. The ratio between the power consumption of the AMOLED screen (typically 237mW to 363mW) and the whole smartphone reduces to between 15% and 22%. Fig. 5 also shows that CPU's generally consume more power than GPU's. In the Galaxy S and S2, the contribution of the CPU to total power consumption can be up to 40%.

5.3 Camera Power Performance

Besides streaming video players and video games, camera recording is another important display-related application on smartphones [17]. As has been studied before, camera recording incurs a surprisingly high power cost. However, the performance with the AMOLED display is not yet evaluated in this regard. In our experiments, we found that:

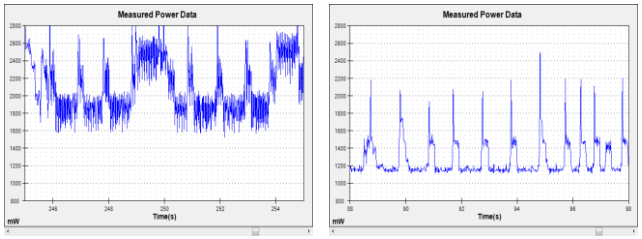
The AMOLED display's power contribution is relatively small compared with that of the rest of the system during camera recording. Possible power reduction may be achieved by optimizing the internal data transformation.

To arrive at this conclusion, we measured the power consumption of camera recording applications. Most of the measured smartphone cameras have a very high resolution of 6 to 8 megapixels and are capable of recording HD videos at up to 1080p. Four test modes are included in our measurements: 1) contrast mode, where the camera works as normal but the AMOLED screen is disabled; 2) preview mode, where we only use camera for preview but not recording; 3) high quality recording, where the video resolution is 1280x720; and 4) low quality recording, where the video resolution is 640x360. To exclude the power variance



(a) (b) (c) (d)

Fig. 6 Galaxy S2 camera recording power histogram: (a) Preview mode; (b) Low quality recording; (c) Zone-in of the power histogram of high quality recording; (d) Zone-in of the power histogram of low quality recording.



(a) (b)

Fig. 7 Galaxy S2 camera recording power histogram: (a) 3264x2448; (b) 320x240.

introduced by the AMOLED screen itself, the smartphone cameras record the same video test benches projected to a screen, including pure black, still picture, and the videos from typical categories. Except for the auto focus, other additional effects are disabled.

A typical power histogram of Galaxy S2 smartphone in preview mode is shown in Fig. 6(a). Note that the power consumption of the AMOLED screen can be derived by subtracting the power utilized during contrast mode from the power consumption in preview mode. In recording modes, the power consumption of the smartphone increases, as shown in Fig. 6(b). The power variances among the different display contents are barely recognized and dominated by the video processing with some impacts from recording quality. For a 4-minute camera recording, the average power consumption is 1400mW and 1650mW with the low- or high-quality recording, respectively. Out of the total power consumption, the portion of the AMOLED screen accounts for 170mW and 180mW in the low- or high-quality recording, respectively. Here the frame rate is 30fps. The power difference incurred by the resolution up-scaling is 250mW=1650mW–1400mW, which is quite small compared to the overall.

However, the zone-ins of the power histograms of each recording mode in Fig. 6(c) and (d) show periodic spike patterns. The highest spikes in the low and high-quality recordings are around 1620mW and 2050mW, respectively. Also, the spike pattern during low-quality is more regular than that in high-quality recording.

We believe the difference of camera spike patterns in the two types of video recordings is introduced by the constraints on data communication. Since the camera buffer capacity is limited, the power consumption due to data transfer in the high-quality recording becomes more apparent when the frame size increases. We also tested the camera recording application with the resolutions of 3264x2448 and 320x240, as shown in Fig. 7. Distinct spike pattern difference are observed in these two tests. As expected, the spike pattern during the 3264x2448 resolution recording becomes prominent.

The power breakdowns of the camera recording application on different smartphone models is shown in Fig. 8. The contributors to the total power consumption include encoding process, camera device, display and base power consumption. Since the display content captured from the real environment generally have a very low luminance, the power consumption of the AMOLED screen is only 200mW to 400mW. The power consumption of video encoding, which is greatly impacted by the internal data communication, varies significantly among the different models.

Hence, with the tested results, we reached the conclusion that the power consumption of the AMOLED display only accounts for a small part of the device's overall power consumption when recording video from the camera.

6. CONCLUSION

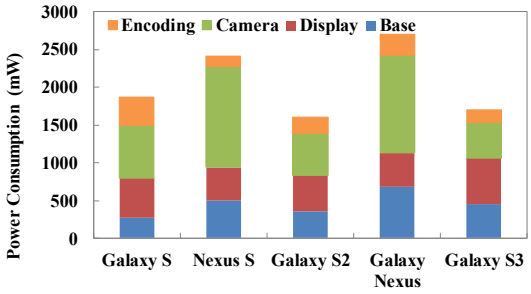


Fig. 8 Power component breakdown in camera recording.

In this paper, we evaluated the power consumption of various display-related applications on smartphones. We first refined the power analysis of AMOLED display by considering design metrics such as sub-pixel area and matrix. We then conducted the AMOLED screen power analysis in the smartphone applications of a streaming video player, video game, and camera recorder. A wide selection of display content was tested during our experiments running on several representative Samsung smartphone platforms. We found that the AMOLED screen power model is heavily affected by sub-pixel matrix design while the power efficiency over the unit area is almost the same. We also found that the power consumption of the AMOLED screen while watching a video is much less than expected while decoding process power consumption is also not significant. In video games, the power consumption of the AMOLED screen varies significantly and power optimization should focus on the CPU side. Finally, camera recording incurs surprisingly high power consumption, which is constrained by the internal data transformation, with the AMOLED display accounting for only a small portion of overall power cost.

7. REFERENCES

[1] J. Huang, *et al*, "Anatomizing Application Performance Differences on Smartphones," *MobiSys*, 2010.
[2] M. Dong, Y. Kevin, L. Zhong, "Power Modeling of Graphical User Interfaces on OLED Displays," *DAC*, 2009.
[3] R. Mittal, A. Kansal, R. Chandra, "Empowering Developers to Estimate App Energy Consumption," *MobiCom*, 2012.
[4] X. Chen, *et al*, "Quality-retaining OLED Dynamic Voltage Scaling for Video Streaming Applications on Mobile Device," *DAC*, 2012.
[5] M. Dong, L. Zhong, "Chameleon: A Color-Adaptive Web Browser for Mobile OLED Displays," *MobiSys*, 2011.
[6] M. Dong, Y. K. Choi, L. Zhong, "Power-Saving Color Transformation of Mobile Graphical User Interfaces on OLED-based Displays," ISLPED, *2009*.
[7] B. Anand, *et al*, "Adaptive Display Power management for Mobile Games", *MobiSys*, 2011.
[8] Android PowerManager, http://developer.android.com/.
[9] PowerTutor, http://powertutor.org/.
[10] C. Yoon, *et al*, "Appscope: Application Energy Metering Framework for Android Smartphone Using Kernel Activity Monitoring," *USENIX ATC*, 2012.
[11] L. Zhang, *et al*, "Accurate Online Power Estimation and Automatic Battery Behavior based Power Model Generation for Smartphones," *CODES/ISSS*, 2010.
[12] AMOLED, http://en.wikipedia.org/wiki/AMOLED/.
[13] Mobile Digital Natural Image Engine (MDNIe), http://ww.samsug.com/.
[14] J. Jacobs, D. Hente, E. Waffenschmidt, "Drivers for OLEDs", *Industry Applied Society Annual Meeting*, 2007.
[15] D. Shin, Y. Kim, N. Chang, M. Pedram, "Dynamic Voltage Scaling of OLED Displays," *DAC*, 2011.
[16] M. Dong, L. Zhong, Z. Deng, "Performance and Power Consumption Characterization of 3D Mobile Games," *Computer*, 2012.
[17] H. Falaki, *et al*, "Diversity in Smartphone Usage", *MobiSys*, 2010.
[18] Samsung Galaxy OLED Display Technology Shoot-Out, http://www.displaymate.com/.

A2PSM: Audio Assisted Wi-Fi Power Saving Mechanism for Smart Devices

Mostafa Uddin
Department of Computer Science
Old Dominion University
Norfolk, VA, USA
muddin@cs.odu.edu

Tamer Nadeem
Department of Computer Science
Old Dominion University
Norfolk, VA, USA
nadeem@cs.odu.edu

ABSTRACT

Wi-Fi is the most prominent wireless network interface in current smart devices. Due to its high power consumption, Power Saving Mode (PSM) schemes have been proposed to reduce power consumption. We show how the current popular PSM schemes implemented in nowadays smart devices are inefficient. In this paper, we propose A2PSM: an audio channel assisted power saving scheme for the Wi-Fi interface, which address the inefficiency of the existing power saving schemes in smart devices. In this scheme, we leverage the low power consumption of the audio interfaces (mic/speaker) to reduce the wakeup events of the Wi-Fi interface when it is in Power Saving Mode. In this paper, we develop a small-scale prototype testbed on real smartphones to evaluate the proposed A2PSM scheme. Experiments show that A2PSM could save up to more than 25% more power than the existing schemes. To the best of our knowledge, this is the first work to utilize the audio channel in optimizing the power consumption of Wi-Fi networks.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design

General Terms

Design, Algorithms, Measurement, Performance

Keywords

WiFi, Constant Awake Mode, Power Saving Mode, Audio Interface

1. INTRODUCTION

Wi-Fi is becoming the prominent network interface for data communication in smart devices (e.g., smartphones) because of its low/free cost, high throughput, relatively large range, and ubiquitous accessibility. However, the WiFi network still has several inefficiencies in terms of high energy consumption, unfairness between co-located nodes, and poor bandwidth utilization. For example, the Wi-Fi transmitter has to finish the transmission of the packet even in the case when the receiver flags the packet as corrupted at early stage of the transmission. Another example is the overhead of the low rate transmission of Wi-Fi control packets (RTS/CTS/ACK).

Addressing the above problems, we introduce the idea of enhancing data communication performance over Wi-Fi networks by using the mic/speaker in smart devices as a parallel communication channel. More specifically, we envision a novel communication framework that utilizes the audio interface (i.e., mic/speaker) on smart devices in developing more efficient Wi-Fi networks. Unlike other interfaces in smart devices such as Bluetooth interface, audio interface (i.e., hardware and software) in current commodity smart devices are open and flexible that enable us to integrate it with Wi-Fi interface at the networking lower layers (i.e., MAC and PHY layers) to realize our vision.

In audio interface, we exploit the frequency band beyond the human ear's perception for audio communication. Nowadays, most of, if not all, smart devices are both capable of generating and discerning audio frequencies beyond the human perception. In this paper we develop and evaluate, as a part of our framework, an audio channel assisted Wi-Fi power saving mechanism for smart devices. To the best of our knowledge, this is the first work to utilize the audio channel in optimizing the power consumption of Wi-Fi networks.

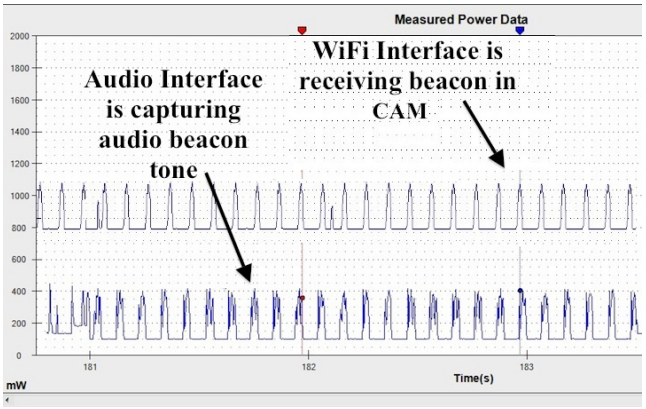


Figure 1: Monsoon Power monitoring result while audio interface is receiving audio beacon tone, and wifi interface is receiving beacon during CAM.

ACM HotMobile'13, February 26–27, 2013, Jekyll Island, Georgia, USA.

2. MOTIVATION AND CONTRIBUTION

Wi-Fi power management has an important impact on the battery endurability of smart devices. In this paper, we propose the Audio Assisted Wi-Fi Power Saving Mode (A2PSM) scheme that exploits both the audio interface hardware (mic/speaker) and the existing Wi-Fi power saving mechanism in smart devices. Given the wide adaptation of Wi-Fi in many types of smart devices and the corresponding spread of peer-to-peer applications and services (e.g., file sharing, multiplayer games, media streaming) between smart devices in home and office setting, Wi-Fi Direct (SoftAP) standard [3, 5] that allows direct communication between Wi-Fi peers is gaining more interests and spread. Wi-Fi Direct and the corresponding power saving mechanism works more like Wi-Fi infrastructure mode in which one device (e.g. smartphone, TV, laptop) acts as an access point (AP) while the others as client stations (STAs). Therefore and without loss of generality, we focus in this paper on using A2PSM in the Wi-Fi infrastructure mode that is equally applicable to Wi-Fi Direct.

Most smart devices have two main power management modes for Wi-Fi Infrastructure mode; Constant Awake Mode (CAM) and Power Saving Mode (PSM). In CAM mode, the device's Wi-Fi interface remains awake all the time while in PSM mode device's Wi-Fi interface awakes periodically to receive beacons from the access point (AP). Because of the importance of power management, several PSM schemes have been applied. The most common used PSM scheme in smart devices is Static PSM (SPSM). In SPSM, the sleeping duration is fixed and set at the association process between the device (STA) and the access point (AP). Since a STA needs to wake up periodically even if there is no data to exchange with the AP, SPSM is not optimal. In our proposed A2PSM, we try to mimic the optimal situation by using the audio interface as a parallel channel to the Wi-Fi interface in order to allow the STAÕs Wi-Fi interface sleep as long as there is no data to exchange between the AP and the STA.

To understand the power consumption of both the audio and Wi-Fi interfaces, we conducted a simple experiment with the use of Monsoon power monitoring tool [2]. In the experiment, the audio interface and the Wi-Fi interface receive an audio beacon (i.e., tone) and a Wi-Fi beacon respectively in a periodic pattern. Clearly, Figure 1 shows that the power consumption of the Wi-Fi interface is three times more than the audio interface. Such observation motivated us to consider a new mechanism to utilize the audio interface in enhancing the current PSM mechanism in the smart device to optimize its power consumption.

In the following, we summarize the contributions of this paper:

1. *Introduce the idea of using the audio communication channel to assist the Wi-Fi PSM mechanism.* This paper takes the first step to explore the feasibility and the future direction of our framework that utilizes the audio interface as an additional interface for the Wi-Fi interface.

2. *Design the A2PSM scheme for smart devices.* We describe in details about the design challenges of incorporating the audio channel with Wi-Fi.

3. *Implement the A2PSM scheme on commodity smart device.* We Implement our prototype of A2PSM on Nokia N900 phones.

4. *Evaluate the A2PSM scheme in real environments.* We evaluate the power efficiency of our implemented prototype using Monsoon power monitoring tool. We also identify several steps for future research.

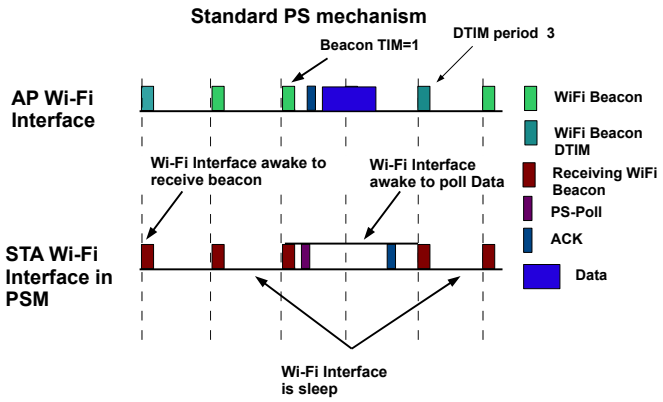


Figure 2: Standard PSM scheme for Wi-Fi infrastructure networks

3. BACKGROUND: PSM OVERVIEW

In standard PSM, the STA's Wi-Fi interface awake periodically to receive beacons from the AP. A beacon message from the AP with the STA's TIM bit set indicates that AP has buffered data for the STA. Consequently, the STA's Wi-Fi interface switches from PSM to CAM and poll the data from AP. In case of a broadcast frame, AP transmits a special kind of beacon message; DTIM, periodically every certain beacon intervals. Figure 2 gives an overview of the standard PSM scheme for Wi-Fi infrastructure networks. Unlike PSM, A2PSM scheme keeps the STAÕs Wi-Fi interface in sleep until there is a data to exchange with the AP and the STA needs to switches from PSM to CAM. While the STA is in sleep, the communication between the AP and the STA happens thru the audio interface. In our schema, we assume both the AP and the STA have an audio interface (mic/speaker). In section 5 we describe details of our A2PSM scheme.

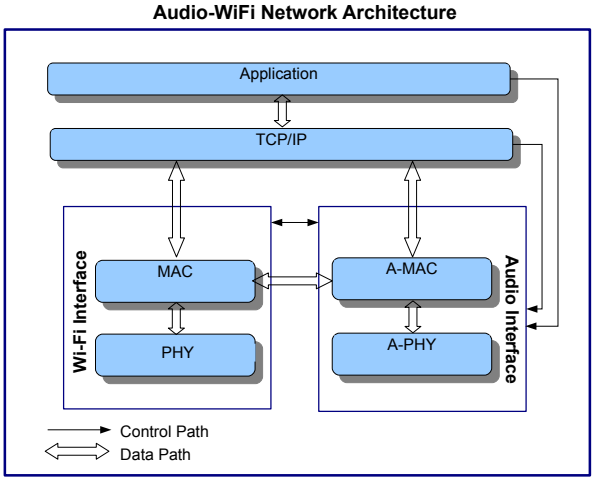


Figure 3: Preliminary Audio-WiFi Framework Architecture.

4. AUDIO-WIFI ARCHITECTURE

Figure 3 shows the preliminary architecture of our Audio-WiFi framework vision. The audio interface in this framework has two layers: 1) A-PHY layer: responsible for signal processing and signal transmission/ reception using speaker/microphone hardware. 2) A-MAC layer: responsible for sending/receiving audio signals over audio channels. In this framework, Wi-Fi MAC and upper layers (e.g., TCP/IP and applications) utilize the audio interface to send control frames (e.g. beacon frame) using the A-MAC layer. In a similar way, A-MAC could receive control frames over audio channel and send it to Wi-Fi MAC and upper layers. In addition, the framework defines the cross-layer interactions (e.g. control path) between the different layers and the audio interface to facilitate the data flows between the different components. In the implementation section we describe, how we use this architecture to implement our prototype of the proposed A2PSM scheme in Linux-based Nokia N900 smartphones.

5. A2PSM DESIGN

The scope of this paper is to design and implement the A2PSM scheme for unicast transmissions in Wi-Fi infrastructure networks. We will discuss the broadcast transmissions later in Section 9. In designing A2PSM scheme, we need to fulfill the following requirements: (1) The use of the audio interface with the PSM should cost less energy than the traditional Wi-Fi interface. (2) The audio interface needs to generate and receive the audio beacon within the same time limit of the Wi-Fi beacon, in order not to increase the data transmission latency. In this section, we describe in details the design of A2PSM scheme that fulfill the above requirements

5.1 Audio Beacon

In A2PSM, in addition to the Wi-Fi beacons, the AP needs to generate audio beacons synchronized with the Wi-Fi beacons. The audio beacon is an audio signal that has one or more high frequency ($\geq$ 18kHz) sinusoidal signal components. In 802.11 standards, the AP assigns a unique Association Id (AID) to each STA during the association process. In A2PSM, similarly, each STA is assigned a unique audio frequency corresponding to its unique AID. Figure 4 shows the overall power saving mechanism of the A2PSM. As shown, when the AP has a buffered data for a STA or more, the AP includes in the audio beacon, transmitted along with the Wi-Fi beacon, the frequency components that are corresponding to these STAs. This audio beacon structure mimics the Wi-Fi beacon with TIM bits. In A2PSM, while the STA is in PSM, the audio interface awakes periodically to capture the audio beacons from the AP. When a STA receives an audio beacon with including its audio frequency, it puts its audio interface to sleep and then awakes the Wi-Fi interface to poll its buffered data from the AP. Note that, the duration of the awake periods of the audio interface should be short enough to capture the audio beacon while minimizing the energy consumption. In our implementation, we set this period to 20ms, which is the average duration of the Wi-Fi interface being awake to receive the Wi-Fi beacon.

5.2 Relative position

In Figure 4, there is a time difference between the generation and the reception of the audio beacon tone by the AP and the STA respectively. This time difference is due to the slow propagation speed of the audio. The observed time difference depends on the relative distance between the STA and the AP. The STA and the AP can use one of the existing ranging schemes such as Beepbeep [10] to estimate the relative distance between them. Knowing the relative distance to the AP, the STA could calculate the exact time to awake its audio interface just right before the reception of the audio beacon. Therefore, STAs with different distances to the AP turn their audio interfaces at different times.For example, in Figure 5, while *STA2* has not received the audio beacon yet, *STA1* started to receive the audio beacon.

To satisfy our second design requirement, the farthest STA from the AP needs to receive the audio beacon within the typical Wi-Fi beacon interval time (e.g. 100ms). Such requirement is necessary to minimize the data latency between STAs and the AP. Devices in Wi-Fi Direct scenarios are in close proximity (e.g. 5-10 meter) to each other. For the indoor Wi-Fi networks (e.g., home and office settings), the typical average Wi-Fi range is within 30 meter [6]. Given the speed of sound in air, the farthest STA would start receiving the audio beacon within 60ms after its transmission for the 30meter range. This allows the farthest STA to capture the audio beacon within the typical Wi-Fi beacon interval (e.g., 100ms). This requirement is more challenging in case of broadcast transmissions. We describe this challenge in section 9.

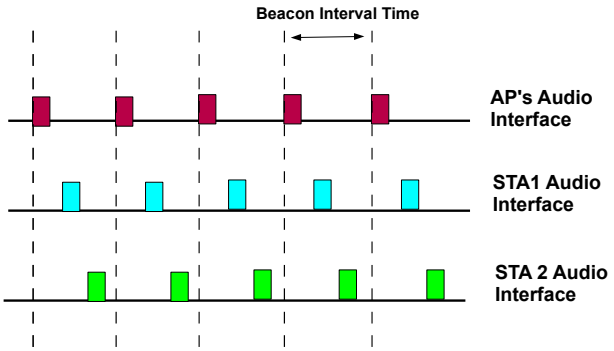


Figure 5: Audio beacon communication between two STAs and AP. In the scenario, STA2 is further away from the AP compare to STA1.

6. IMPLEMENTATION

We have implemented our prototype of A2PSM on two Nokia N900 smartphones. We use one phone as an AP and the other as a STA. In the prototype, we implement the followings: (1) The audio interface for both the STA and the AP. (2) The interaction between the Wi-Fi Interface and the audio Interface for both the STA and the AP.

6.1 Audio Interface

In the audio interface, we implement a simple prototype of the A-PHY layer. Figure 6 shows the original components of the Linux sound driver (e.g. ALSA) and the Wi-Fi driver(e.g. wl12xx) (shown as dotted boxes) in Linux-based smart devices (e.g. Android, Memo etc.). Our additional A-PHY and A-MAC modules of the audio interface are shown in the figure as shaded solid boxes. The A-PHY of the audio interface is responsible to transmit/receive the audio beacon. In our implementation, the A-PHY module in the AP only generates the audio beacon synchronized with the Wi-Fi

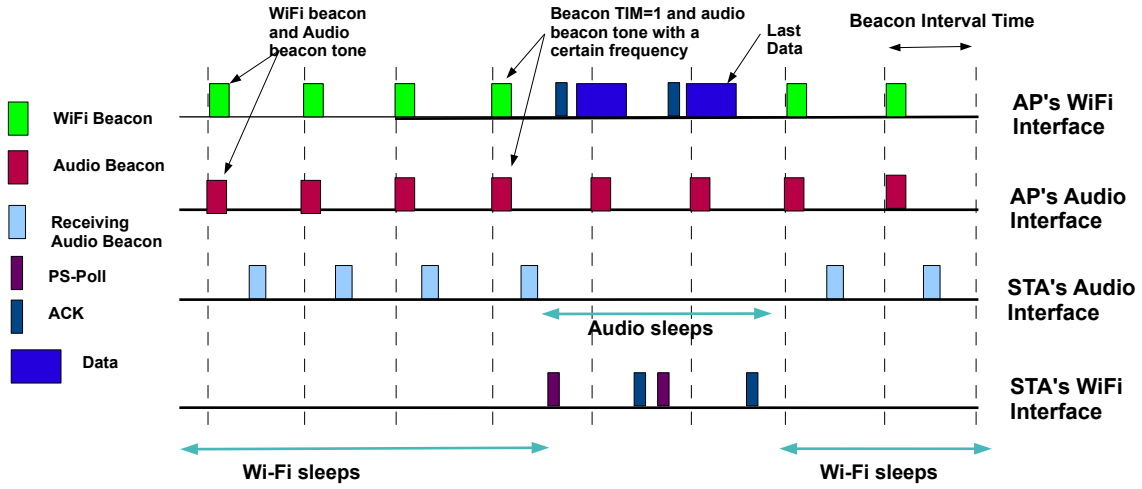


Figure 4: Overview of unicast transmissions from an AP to a STA using A2PSM scheme.

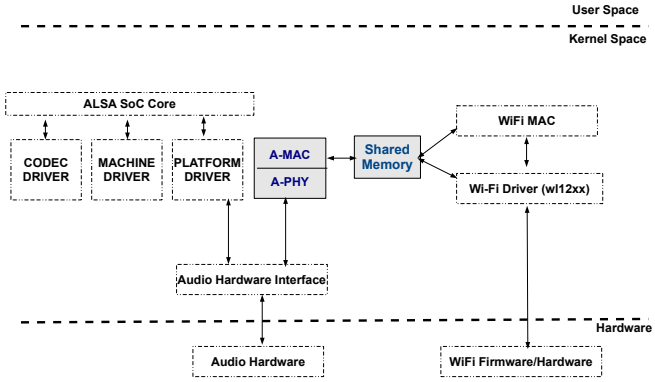


Figure 6: Implementation architecture of A2PSM scheme.

beacon transmission. On the other hand, the A-PHY module in the STA is responsible to capture the audio beacon and to detect whether its corresponding frequency component is included in the beacon. Initially, the A-PHY module in the STA applies a high pass filter to extract only the high frequency component of the captured audio signal. Followed by the filtering, the A-PHY module uses cross-correlation technique to detect whether the captured signal contains a certain frequency component corresponding to the STA's AID. Given that the background noises have very minimal effects in the high frequency range (18kHz-21kHz), it is easy to detect certain frequency components within the captured beacon with high accuracy.

The A-MAC module in the AP interacts with the A-PHY module to transmit periodic audio beacons aligned with the Wi-Fi beacons. In addition, the A-MAC module is responsible to define the frequency components in the audio beacon based on the AIDs of the STAs those have buffered data at the AP. Therefore, the A-MAC module in AP needs to communicate with the Wi-Fi MAC to identify those STAs. In the STA, the A-MAC module notifies the Wi-Fi MAC only when it has data to poll from the AP.

One main question we need to address in our implementation is: how long we need to keep the audio interface awake in the STA to capture the audio beacon properly? Our objective in the implementation is to minimize the duration needed by the audio interface in the STA to stay awake to capture the audio beacon. Since the duration of capturing an audio beacon at the STA is tightly coupled with the duration of the generated audio signal at the AP, we studied how audio signal is generated. In general, the generation of an audio signal depends on three main parameters of the sound driver; sampling period, buffer size and period size [8]. In our implementation, we tune these parameters to limit the duration of the audio beacon signal to 10ms, which is large enough to be detected at the STA side. Therefore, the audio interface at the STA needs to keep awake for at least 10ms in order to capture the audio beacon from the AP. This duration period is identical to the period the Wi-Fi interface stays awake in the SPSM scheme to receive a Wi-Fi beacon.

6.2 Audio-WiFi Interaction

Another important issue we have to address in our implementation is to minimize the miss-alignment between the transmissions time of the audio beacons and the WiFi beacons at the AP. In doing this, fast interaction between the audio and the Wi-Fi interfaces is needed to guarantee lower miss-alignment between the audio and Wi-Fi beacons. In our implementation, we use a simple shared memory to facilitate the interaction between the two interfaces thru sharing certain information that control the operation of the interfaces. For example, the Wi-Fi driver uses the shared memory to signal the audio driver about a Wi-Fi beacon transmission. Similarly, the Wi-Fi drive uses the shared memory to inform the audio driver about changing its PSM mode.

7. EXPERIMENT AND EVALUATION

In this section we evaluate the performance of our proposed A2PSM scheme by answering the following two questions: (1) How much energy is saved by the A2PSM scheme compared to the standard power saving mechanism under different traffic loads? and (2) Does the A2PSM has any effect on the network throughput?

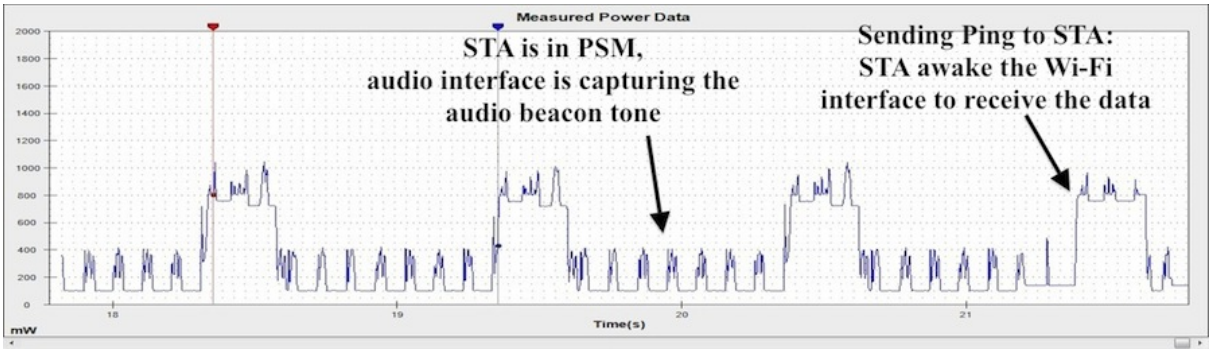


Figure 7: Monsoon Power Monitoring observation of A2PSM schema while STA is receiving ping message during PSM.

In our experiments, we use two Nokia N900 phones running our implemented A2PSM scheme in which one phone act as a STA and the other as an AP. We connect the STA with the Monsoon Solution Power Monitor [2] to measure its energy consumption. We conducted a simple experiment to validate the proper implementation of the A2PSM. In this experiment, we send a periodic ping command through the AP to the STA. Figure 7 shows the power consumption that is corresponding to the activities of the audio and Wi-Fi interfaces in the STA in response to the ping commands. This plot validates the proper operation of A2PSM scheme in receiving the audio beacon and in switching between the audio interface and the Wi-Fi interface.

To evaluate the energy efficiency of A2PSM scheme, we run iperf [1] tool on both the STA and the AP in server mode and client mode respectively. We send UDP data from iperf client to the iperf server under different traffic loads for duration of 20 seconds. We replicate this experiment 10 times for each traffic load setting. During the experiment, we fix the distance between the STA and the AP to 3 meter. We compare the power consumption of the smartphone under two different power saving schemes: A2PSM and SPSM. During the power consumption measurement, we turn off all the other radio interfaces as well as the display screen of the smartphone. Moreover, we use the same settings of the smartphones under both power saving schemes.

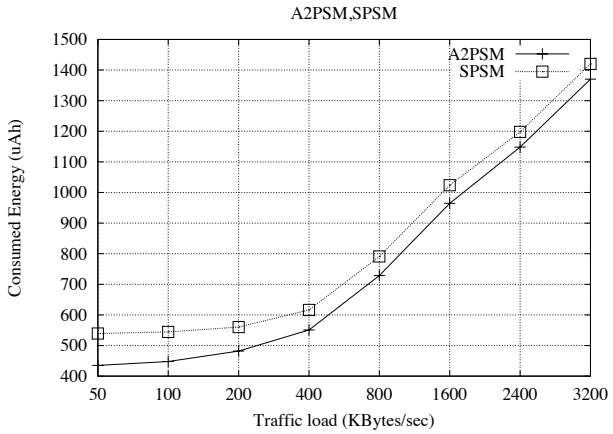


Figure 8: Power consumption comparison between A2PSM and SPSM schemes under different traffic loads.

Figure 8 shows the power consumption for both SPSM and A2PSM schemes under different traffic loads. Each point in the figure is the measured power consumption of the STA averaged over 10 experiment runs for the same traffic load. As shown, our implemented A2PSM prototype could save up to more than 25% more power than the existing SPSM scheme. We observe that A2PSM saves more energy at lower traffic loads. For example, A2PSM saves almost 25% more power under a traffic load of 100 KBytes/sec, while it saves only 5% more power under 3200 KBytes/sec load. Figure 9 shows the expected battery life time of the smartphone for both power saving schemes under different traffic loads.

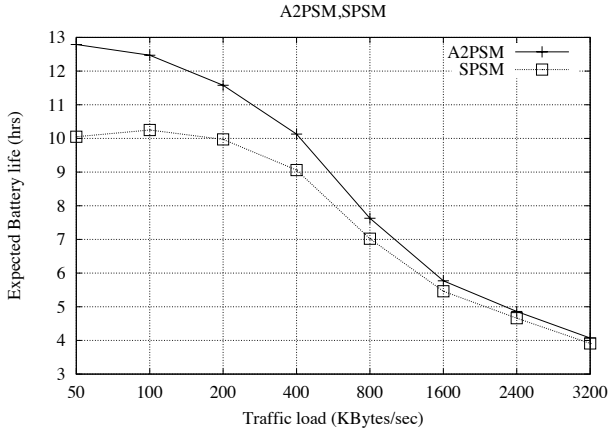


Figure 9: Expected battery life time comparison between A2PSM and SPSM under different traffic loads.

In order to evaluate whether A2PSM scheme has any effect on the network performance, we conduct similar experiment to measure the network throughput. After repeating the experiment several times under different traffic loads, we found out that there is no significant difference in network throughput under both A2PSM and SPSM schemes. Such result validates that our A2PSM scheme has no effect on the network performance.

8. RELATED WORK

Number of prior solutions have been proposed to reduce the energy consumption of the wireless network in smart devices. In

this section, we focus on only those that are related to multiple radio interfaces. Using multiple radio interfaces is a popular idea in optimizing or enhancing the overall performance of the wireless communication [12, 4].

Recently, large focus has been given to improving the power consumption of wireless communication in smart devices. Several proposed solutions focused on utilizing the Bluetooth interface for improvising the performance of the Wi-Fi [11, 4]. For example, CoolSpots [11] enables automatic switching mechanism between multiple wireless interfaces (i.e., WiFi and Bluetooth) of the mobile device in order to extend the battery lifetime. The Blue-Fi system [4] utilizes the nearby bluetooth contact-patterns and cell-tower information to predict the availability of the Wi-Fi connection. Thus, it allows the device to turn off the Wi-Fi interface and minimize the number of idle state periods and and the number of neighbor discovery scans. However, one of the major challenges to be addressed in using Bluetooth with Wi-Fi is the co-existence [7].

Researchers have proposed to use other wireless network interface such as ZigBee as a parallel communication to enhance the energy efficiency of the Wi-Fi network, such as Wi-Zi Cloud [9]. Typically, current smart devices do not support ZigBee interface. Authors in [13] propose to switch and use a separate low-power control channel in addition to the main high-power data channel. When the device is inactive, it turns off the high-power channel and uses the low-power control channel to send control messages (i.e., wakeup message). In this scheme, authors use special hardware as a separate low-power interface in addition to the typical Wi-Fi interface. To the best of our knowledge, A2PSM is the first of its kind that utilizes the audio interface to reduce the power consumption of Wi-Fi interface in smart devices.

9. CHALLENGES AND FUTURE WORK

Broadcast Frame: In case of receiving unicast transmissions, data are polled by STAs from the AP. On contrary, data are pushed by the AP for the broadcast transmissions. This scenario imposes additional challenges in designing A2PSM for broadcast transmissions. Since different STAs using A2PSM scheme will receive the audio beacon at different times as discussed in section 5.2, the AP in broadcast transmissions needs to wait until the farthest STA receives the audio beacon before sending the broadcast data. As a result, STAs that are nearer to the AP will have to wait longer time before receiving the broadcast data. As a consequence, a STA closer to the AP consumes more power in receiving a broadcast data. In future, we plan to address the broadcast transmissions challenge within A2PSM.

Multiple STAs: In this paper we have just implemented a small scale prototype testbed of our A2PSM scheme for one AP and one STA. In future work, we plan to develop a complete A2PSM scheme that supports multiple STAs.

In conclusion, this paper proposes a novel power saving scheme that utilizes both the audio and the Wi-Fi interfaces in smart devices. In the paper, we designed, implemented and evaluated a preliminary version of A2PSM scheme on real testbed. Results are promising and motivate us to continue developing the complete A2PSM scheme for smart devices.

10. REFERENCES

[1] iperf. `http://iperf.sourceforge.net`.

[2] Monsoon power monitoring device. `http://www.msoon.com/LabEquipment/PowerMonitor/`.

[3] W.-F. Alliance. P2p technical group, wi-fi peer-to-peer (p2p) technical specification v1.0, 2009.

[4] G. Ananthanarayanan and I. Stoica. Blue-fi: enhancing wi-fi performance using bluetooth signals. In *Proceedings of The 7th International Conference on Mobile Systems, Applications and Services (MobiSys '09)*, Krakow, Poland, 2009.

[5] D. Camps-Mur, A. Garcia-Saavedra, and P. Serrano. Device to device communications with wifi direct: overview and experimentation. In *IEEE Wireless Communications Magazine*, 2012.

[6] Y. Chen, D. Lymberopoulos, J. Liu, and B. Priyantha. Fm-based indoor localization. In *Proceedings of The 10th International Conference on Mobile Systems, Applications and Services (MobiSys Õ12)*, Lake District, UK, 2012.

[7] R. Chokshi. Yes! wi-fi and bluetooth can coexist in handheld devices. Technical report, Emerging and Embedded Business Unit, Marvell Semiconductor, Inc., March 2010.

[8] S. Dimitrov and S. Serafin. Audio arduino - an alsa (advanced linux sound architecture) audio driver for ftdi-based arduinos. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME '11)*, 2011.

[9] T. Jin, G. Noubir, and B. Sheng. Wizi-cloud: Application-transparent dual zigbee-wifi radios for low power internet access. In *Proceedings of The 30th IEEE International Conference on Computer Communications (INFOCOM '11)*, Shanghai, China, 2011.

[10] C. Peng, G. Shen, Y. Zhang, Y. Li, and K. Tan. Beepbeep: A high accuracy acoustic ranging system using cots mobile devices. In *Proceedings of The 5th ACM Conference on Embedded Networked Sensor Systems (SenSys '07)*, Sydney, Australia, November 2007.

[11] T. Pering, Y. Agarwal, R. Gupta, and R. Want. Coolspots: Reducing the power consumpton of wireless mobile devices with multiple radio interfaces. In *Proceedings of The 4th International Conference on Mobile Systems, Applications and Services (MobiSys '06)*, Uppsala, Sweden, 2006.

[12] H. Qin, Y. Wang, and W. Zhang. Zigbee-assisted wifi transmission for multi-interface mobile devices. In *Proceedings of The 8th Annual International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous '11)*, Copenhagen, Denmark, December 2011.

[13] E. Shih, P. Bahl, and M. J. Sinclair. Wake on wireless: an event driven energy saving strategy for battery operated devices. In *Proceedings of The 8th Annual International Conference on Mobile Computing and Networking (MobiCom '02)*, Atlanta, Georgia, USA, 2002.

Application Modes: A Narrow Interface for End-User Power Management in Mobile Devices

Marcelo Martins Rodrigo Fonseca
Brown University

ABSTRACT

Achieving perfect power proportionality in current mobile devices is not enough to prevent users from running out of battery. Given a limited power budget, we need to control active power usage, and there needs to be a prioritization of activities. In the late 1990s, Flinn and Satyanarayanan showed significant energy savings using a concept of data fidelity to drive mobile application adaptation, informed by the battery lifetime desired by the user and the OS's evaluation of energy supply and demand. In this paper we revisit and expand this approach, recognizing that with current hardware there are even higher potential savings, and that increased diversity in applications, devices, and user preferences requires a new way to involve the user to maximize their utility. We propose Application Modes, a new abstraction and a narrow interface between applications and the OS that allows for a separation of concerns between the application, the OS, and the user. Application Modes are well suited to eliciting user preferences when these depend on multiple dimensions, and can vary between users, time, and context. Applications declare modes – bundles of functionality for graceful degradation when resource-limited. The OS uses these modes as the granularity at which to profile and predict energy usage, without having to understand their semantics. It can combine these predictions with application-provided descriptions, exposing to the user only the high-level trade-offs that they need to know about, between battery lifetime and functionality.

1. INTRODUCTION

Battery life has been a fundamental limitation in mobile devices for as long as they have existed, despite a vast body of literature on power management extending back almost two decades (§6). In fact, increasingly demanding applications greatly exceed the average power draw that would be required for batteries to last through a typical charging period [9, 25].

There is a wide spectrum of proposed solutions for power management. A first class of solutions deals with the management of idle-resource power, by automatically switching hardware components to low-power states when not in use. These include timeout policies for hibernation, suspending disks, displays and radios; and CPU voltage and frequency scaling. The outcome, if these are perfect, is an energy-proportional system [5]. Although necessary, these are not sufficient to solve the increasing energy-deficit problem, because they have no effect when there is *active demand* for resources.

To reduce the active demand, there must be a prioritization of functionality. In the late 1990s, Ellis [8, 27] recognized that the offered workload has to be changed by user-driven prioritization and lifetime goals, and Flinn and Satyanarayanan established, with the Odyssey system, that application adaptation can provide substantial energy gains [10, 16]. In Odyssey, applications automatically and dynamically change their behavior to limit their energy consumption and achieve user-specified battery lifetime, guided by the operating system. The adaptation involves a trade-off between energy use and application data quality, which they called *fidelity*. Fidelity is application-specific and opaque to the OS. The role of the OS is to direct the adaptation based on its evaluation of the supply and demand of energy, and their relation to the expected battery duration. When the OS detects the lifetime goal as unachievable, it issues upcalls to applications so they reduce their fidelity. The user inputs two pieces of information: the desired lifetime, and a prioritization of applications to order their adaptation, whereas application developers are responsible for implementing different fidelity levels.

Flinn and Satyanarayanan were the first to simultaneously involve the OS, the applications, and the user in power management, and many factors in today's environment make it opportune to revisit and extend their approach, which we do in this paper. Due to a combination of more complex applications, multiple devices, and a diverse user base, in some cases there is no single fidelity metric that is common to all users in all contexts, making automated approaches to adapt some applications ineffective. Furthermore, given advances in hardware and in lower-level software (*e.g.*, ACPI), devices are much more efficient when idle, making higher-level approaches that reduce active demand much more effective now than a decade ago.

In [10], as in [12], a fundamental assumption is that there is a well-defined trade-off between fidelity (or QoS) and energy use. This means that an application developer knows the app configurations that lie in the *Pareto frontier* of this trade-off, enabling an automated algorithm to decide the state based on the available energy.

Even though this still holds for many applications, this is not always true. As we show in §2, two users with different preferences can have very different trade-offs between energy usage and *utility* from an application. The key observation is that in these cases, automated adaptation fails, and the runtime system must elicit preferences from the user. The main challenge is how to involve the user at the right time and *at the right level*. She should only worry about tangible aspects of the device operation, such as *lifetime* and *functionality*, and not be concerned with how these are implemented or achieved.

ACM HotMobile'13, February 26–27, 2013, Jekyll Island, Georgia, USA.

In this paper we propose *Application Modes* (§4), an interface between applications and the OS that eases this communication. Rather than exposing a *metric*, applications declare to the OS one or more *modes*, which comprise reductions of functionality with presumed power savings. Modes carry a human-readable description of the resulting functionality, and the promise of switching when requested by the OS. Similarly to previous works [10, 2], we assume that the OS can predict how long the device will last with the application in each mode, and then request its change when appropriate. However, recognizing that different modes may have different utilities for different users, the decision of when to switch modes involves the user when necessary, by combining the description provided by the application with the predictions of change in lifetime provided by the OS.

2. MOTIVATION

In this section we use power measurements with two common smartphone applications — a navigation and a video-recording application — to illustrate two main points. First, we confirm and extend earlier findings by Flinn and Satyanarayanan [10] demonstrating how changes in application behavior can substantially affect energy consumption. Second, using the video-recording application, we show that different users can have very different Pareto frontiers in the utility-energy trade-off, making globally automated decisions ineffective to maximize utility.

We measure the power draw of running these applications in very different *modes*, or bundles of settings. We did our measurements on a Samsung Galaxy Nexus running CyanogenMod ICS 4.1.0. We measured the power draw of the entire phone connecting a Monsoon power monitor to the smartphone battery. To discriminate the energy consumed due to application, we first measured the energy consumed by the phone in the idle state, *i.e.*, not running any applications apart from the base system, and established two baselines with the screen on and off. We kept the screen brightness to the same level for all runs where the screen was on. For navigation, we downloaded data using the 3G data connection when needed, and for the recording application, we used the WiFi network for data upload.

Navigation System Turn-by-turn navigation exercises several hardware resources, including the CPU, GPU, audio, networking, and GPS. It is used in sometimes critical situations, when there is little battery left and the user is in need of orientation to arrive at her destination (and a charging opportunity). There are also interesting trade-offs in functionality, utility, and energy use, depending on which subset of resources the application uses.

We demonstrate potential savings from running Osmand[1] and Navfree[2], two turn-by-turn navigation applications for Android devices with online/offline features, and modifying their settings. We consider five modes, listed in Table 1, from selected parameters for the screen and audio outputs, map-data source and routing. These settings are not transparent to the user, and make specific trade-offs between accuracy and resource usage for a given route. Notable differences between settings include the use of previously downloaded vector maps instead of online tile data, disabling the display and using only audio for directions, and downloading directions for the user to write down! We compare the power draw of calculating and outputting the directions for a fixed, four-mile route.

Figure 1 shows, for each mode, the distribution of instantaneous power-draw samples from the device over the entire experiment. The "Full Features" mode yields a richer trajectory, including extra

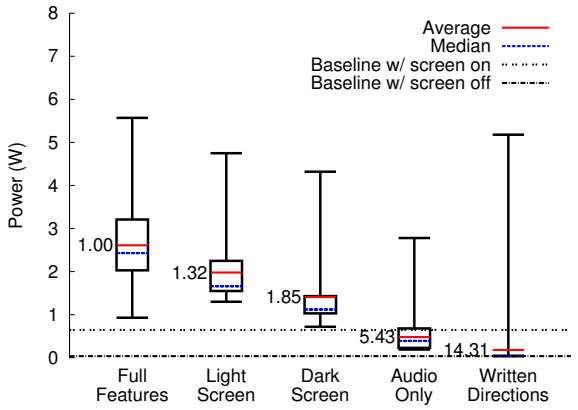


Figure 1: Power draw distributions for the navigation app in different output/routing settings (*cf.* Table 1). The vertical bars show the maximum and minimum power draw, and the boxes the 1st and 3rd quartiles, with the median and average indicated. The number to the left of each bar shows the improvement in battery usage relative to the "Full Features" scenario. Greater energy savings can be achieved by reducing the output quality.

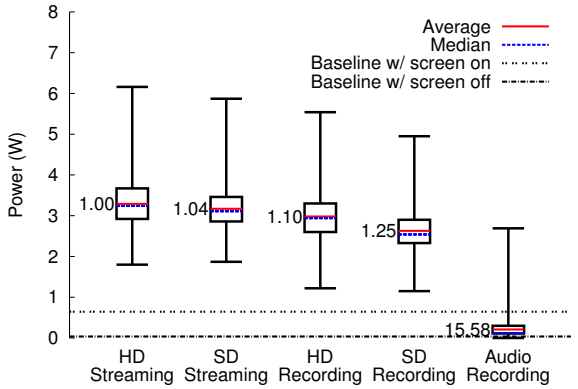


Figure 2: Power draw of different modes for the media-streaming app (*cf.* Table 2).

information like points of interest (POI), at the expense of a larger power profile. As we reduce the number of enabled settings we can see a drop in energy expenditure along with a decrease in the quality of routing information. The "Written Directions" mode draws on average more than 14× less power than "Full Features". The former's high variance stems from briefly using the screen and radio to search for directions; yet, its average power draw is much smaller than its counterparts. In exchange, the user has to take notes of the route before the trip and use them as the only source of information to reach the destination[3]. The potential savings are very significant, provided the user accepts the decrease in quality. In this example, like the ones in previous works, there is a total order in the utility of the modes that is likely agreeable to all users. As such, two alternatives for adaptation can work: as in Odyssey, if the OS knows the user's expected lifetime, the OS can request an increase or decrease in fidelity. Alternatively, we can use Application Modes to expose to users the functionality and expected lifetime of the device in each mode, for them to choose.

[1]http://osmand.net
[2]http://www.navmii.com

[3]This mode was motivated by one of the authors actually having had to do this one time!

Mode Name	Display Settings	Routing Settings	Program Used
Full	Online map tiles and overlays (Mapnik), POIs, compass, polygons	CloudMade routing (online)	Osmand
Light Screen	Offline vector maps, no POIs, no compass, polygons, day mode (light screen)	Osmand routing (offline)	Osmand
Dark Screen	Offline vector maps, no POIs, no compass, no polygons, night mode (dark screen)	Osmand routing (offline)	Osmand
Audio Only	Screen off	Navfree routing (offline)	Navfree
Written Directions	Screen on for browser search, off afterwards, no audio	Google Maps (online)	Browser

Table 1: Alternatives to navigating a four-mile course for the navigation app. Upper modes yield higher-quality routes in exchange for greater resource usage.

Mode Name	Encoding Settings	File size (MB)	Stream transmission?	Program Used
HD Streaming	720p video, hi-def audio	158.60	Live streaming via RTSP	LiveStream
HD Recording	720p video, hi-def audio	290.1	Upload when recharging	SpyCam
SD Streaming	480p video, med-def audio	47.16	Live streaming via RTSP	LiveStream
SD Recording	480p video, med-def audio	183.4	Upload when recharging	SpyCam
Audio Recording	Audio only, screen off	0.58	Upload when recharging	Sound Recorder

Table 2: Functionality alternatives for the media-streaming app, varying encoding quality and immediacy.

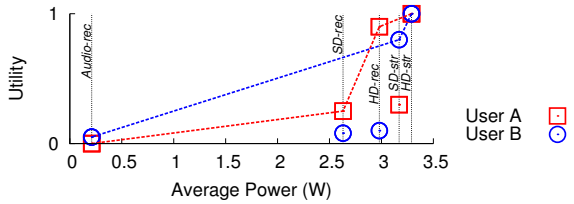


Figure 3: Utilities for the streaming modes for two users. 'A' prefers high-quality capture, whether streamed or recorded, whereas 'B' values immediacy over quality. The Pareto frontiers (the dashed lines) are different, and no single policy can choose between "SD Streaming" and "HD Recording" for all users.

Media Streaming Our second example is an audio and video streaming application which, as navigation, is widely used, leverages different hardware resources, and has interesting trade-offs.

We consider the power draw of capturing and transmitting a five-minute video feed using three similar applications. We performed measurements on different settings of the LiveStream, SpyCam[4] and MIUI Sound Recorder[5] apps using the aforementioned setup. We modify their functionality by selecting parameters for the video and audio encoding and for when to upload the captured media. We consider five basic sets of settings (Table 2), choosing whether to stream or record for later upload, and whether to encode video in high definition, standard definition, or audio only. Once again, these settings are not transparent to the user, and make specific trade-offs between quality and timeliness of the uploaded media.

Figure 2 shows the power draw of each mode. The "Audio Recording" mode draws on average over 15× less power than the "HD Streaming" mode. "Audio Recording" generates the least number of bytes, does not use the screen, camera, or video-encoding hardware, and does not include the transmission energy, as this is done only when the device is recharging.

Figure 3 shows the same modes, with a numerical utility for hypothetical users (which could even be the same user in different contexts). User 'A' is interested in obtaining high-quality video, whereas user 'B' values immediacy. The graph shows that the Pareto frontiers for the two users are very different, and that there is no consistent ordering of the modes, particularly between HD recording and SD streaming. This example highlights that neither the OS nor the application can know a priori the utility of the modes for each user. In this case, and in general when there are multiple dimensions that different users value differently, the automatic selection of a mode breaks down.

[4]http://dooblou.blogspot.com
[5]http://github.com/MiCode/SoundRecorder

3. THE USER NEEDS TO DRIVE

In this section we argue why the user, the applications, and the OS must all be involved in limiting the active demand of mobile devices to achieve maximum *value* out of a limited energy budget.

1. The OS cannot always know the resource priorities of all applications. If an application is consuming too much energy, the OS could limit the resources offered to it, such as CPU time, bandwidth, or access to precise location. Robust applications should sustain such reductions and adapt. However, such arbitrary reductions can be frustrating to the user, *as the value of different functionalities to her may be hard to predict*. This is exacerbated when there are alternative reductions. If the OS decides that a videoconferencing application is spending too much energy, it could reduce its CPU or network allocation, but cannot know which will lead to a more acceptable degradation to the user.

2. Applications cannot always know the functionality priorities of the end-user. Applications are in a better position than the OS to make such decisions, but they may still not know the user's preferences. As the video example in the previous section highlights, there may be no total ordering of the modes in an application, so that the application developer cannot determine the modes in the Pareto frontier for a specific user. In this case, it is only the user who can determine the relative value of the alternatives, as just knowing the user's desired battery lifetime is not sufficient to maximize the utility automatically.

3. Users should choose at the right level, trading off functionality versus lifetime. Although many existing systems could involve the user, most require too much knowledge at the wrong level of abstraction. The user should only have to know about high-level functionality and device lifetime, and not be concerned about which components of the phone even exist. A user wanting her battery to survive a 12-hour flight should not need to understand or even specify the screen brightness, CPU frequency, scheduling algorithm, or the WiFi data rate of her smartphone to fulfill her needs. The phone should hide these trade-offs from the user whenever possible. Popular solutions for end-user energy management are based on *components* rather than functionality, requiring the user to know the resource implications of turning off 3G, GPS, synchronization, or Bluetooth. Frameworks like Cinder [19], Koala [21], and Chameleon [14] have mechanisms to limit resource usage per application, but suffer from the same problem – they assume mobile users are likely to become system administrators or programmers of their devices. On the other hand, other frameworks limit themselves to a single knob, such as lifetime [10, 27] or a preference between performance and duration [1], but as we show, in some cases, this is not enough to maximize utility.

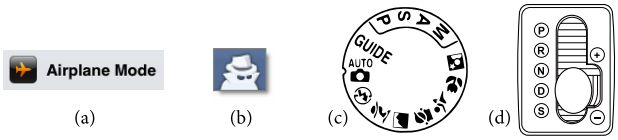


Figure 4: *Application Modes* abstract complex settings by a single, functionality-centered concept, and are common in diverse settings: (a) Airplane mode on phones, (b) Incognito mode in Chrome, (c) scene modes on a camera, and (d) driving modes on a semi-automatic transmission.

The only remaining question is why the OS should be involved at all, since applications could elicit users' preferences directly. The challenge here lies in the decision of when apps would offer these choices, as this requires knowledge of current and future energy availability. This functionality may require device-specific modeling, and should more naturally reside in the OS [6]. Requiring it in each app entails duplicated developer effort, and leads at best to poor or inconsistent results. The OS, on the other hand, is in the right position to provide an energy context to all apps, including profiling and predictions of energy usage and lifetime.

4. APPLICATION MODES

To address the concerns in the previous sections, we have implemented a new abstraction named *Application Modes*, bundles of functionality declared by applications to ease the separation of concerns required between applications, the user, and the OS for effective resource management. We borrow the concept of modes from several commonplace settings (see Figure 4), where they represent complex settings abstracted by a single functionality-centered, easy-to-understand concept. Application Modes resemble Chroma [2] in that very little application knowledge is exposed. Different from Chroma, users are not oblivious to the decisions made by the system, but actually have an active voice in making the decisions that affect their experience. Application Modes are particularly well suited to cases where there are multiple dimensions involved in the users' implicit preference function, with no total order among them, similarly to the different shooting modes on a camera, for example.

Power savings are achieved through *graceful degradation*. Developers create different modes for an application by selecting sets of functionalities that entail different application behaviors, as perceived by the user, in exchange for reduced energy consumption. Graceful degradation is achieved through various ways: different settings, different algorithms [20], even different programs.

The central part of the abstraction is a narrow interface between applications and the OS (see Listing 1). When opened for the first time, applications implementing this interface via a shared library declare to the OS their supported modes using a label and a user-friendly description of how each mode affects the user experience (the registerModes() system call). An OS-listening component intercepts this system call and saves in its database the metadata passed as arguments by the application, along with a unique identifier for each mode, and the currently selected mode. The latter is necessary to automatically restore the behavior of applications once they are reopened. Applications supporting our abstraction promise to switch to a given mode when instructed by the OS, whereas applications oblivious to this new API are not affected. Table 3 lists a few mode examples for different apps.

```
registerModes(List<ModeData>);   // system call
setMode(ModeId);                 // callback
```

Listing 1: API for data exchange between applications and the OS.

Application Modes represent a meaningful granularity at which the OS can do profiling of energy consumption, and make lifetime predictions for each one. Applications keep control of what resources to reduce in order to save power, but leave the decision of *when* to do so to the OS, which has detailed knowledge of the energy context of the entire device, and to the user, who can prioritize application functions based on her goals.

Battery lifetime and high-level functionality, metrics understandable by *both* users and developers, are used to guide adaptation. In one possible scenario, the OS notices that, at the current power draw, her phone will exhaust the battery before the next expected recharge. The OS then presents to the user a list of running apps, ordered by power draw. When the user selects an app from the list, the OS presents an interface similar to that in Figure 5, on which the user selects a different mode for the app, informed by its description (*i.e.*, functionality) and expected impact on battery lifetime. In another usage scenario, the user is presented with a notification of Application Modes support when opening a new program that implements the API. After clicking the notification, an interface similar to Figure 5 appears, and she explores the trade-off possibilities. Once a mode is selected, the OS instructs the application to change its settings using the setMode() callback function. It is the responsibility of the application developer to instruct her program to correctly change its behavior according to the mode selection.

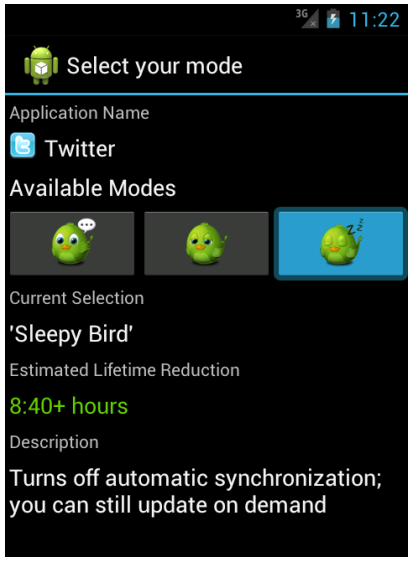


Figure 5: Interface to select application modes for a Twitter app.

5. CHALLENGES

For Application Modes to be adopted and maximize energy savings and user satisfaction, we and the community need to address a number of important challenges.

Energy Profiling and Forecasting The OS is at the right place to maintain an energy context for the device. This includes profiling the energy consumption due to apps, and forecasting the expected battery life given the current and alternative settings. Profiling at least at the granularity of Application Modes is needed for guiding developers on how to choose and optimize the modes themselves, and forecasting is key in telling the user the impact of choosing different modes. There has been significant progress in this area [10, 11, 15, 18, 24, 28], but there is still room for more precision, incorporation of segmented data from large user populations ([17] is a great start), and better support for sharing and delegating resource usage.

Developer Adoption Application Modes place a burden on developers, and are only useful if adopted. We are hopeful that there will be enough interest, given the benefits and the fact that some apps al-

Application	Full	Medium	Powersaver
Location Tracking	1m precision, real-time	1m precision, every 15 minutes	50m precision, at least once a day
Navigation	3D map, audio, real-time location	3D map, audio, location near turns	2D map with directions only
Video Upload	HD video, real-time upload	SD video, real-time upload	SD video, upload when charged
Twitter	Real-time updates	Updates every 5 minutes	Updates on demand

Table 3: Example modes for some applications.

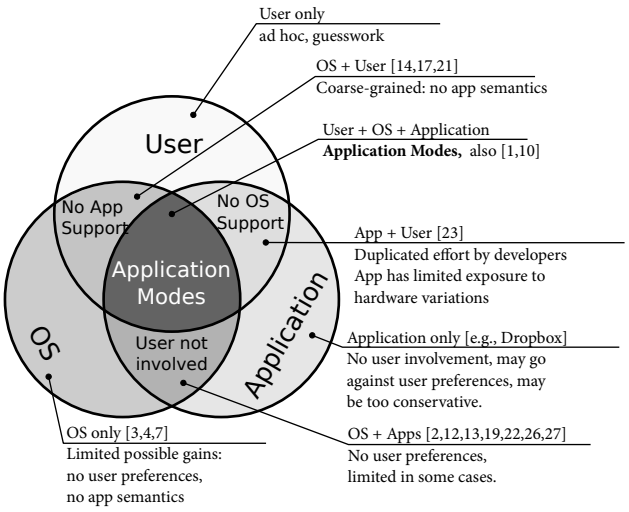


Figure 6: Design space of power-management solutions based on whether there is involvement or support from the user, the application, or from the OS. Application Modes elicit preferences from the user, using lifetime predictions from the OS, and functionality descriptions from the application.

ready change their behavior in response to the context. Dropbox on Android, for example, has an automatic photo-upload option, and disables the feature if the battery is low. With Application Modes, the developer would not have to write code to make sense of the battery context. As per the previous paragraph, developers need accessible energy profiling, as it is only by measuring the impact of design decisions that a developer will make informed decisions to effectively create modes. Lastly, the developer needs guidelines to not create too many modes, modes that do not affect perceived app behavior or expose too much detail. As in Chroma [2], we expect the number of *relevant* energy-saving optimizations from applications to be small.

Limiting User Involvement Changes to the end-user experience should be as little intrusive as possible. While user input is required in many situations, the OS and applications should autonomously act upon energy-related decisions as much as possible, through models, services, and profiles, only escalating what is really important.

Conflict Detection and Resolution Since most current mobile devices support multitasking, two or more apps could have conflicting modes. If two apps use the radio, having only one of them promising not to is of no use. The OS should have a mechanism to detect and resolve such conflicts, with possible involvement of applications and ultimately of end-users. Another source of conflicts are global settings not associated with any apps: a user setting the screen brightness is one example.

Other Resources The concept of Application Modes may apply to contexts other than energy, such as data usage and privacy. Related to previous challenges, the interaction and potential conflicts of these modes, and the possibility of an explosion of their number are important challenges we should address.

6. RELATED WORK

We build upon a large body of previous work that explores support from subsets of the user, the application, and/or the OS for power management. We structure our discussion around Figure 6, which lays out this space and points to representative works on each subset. Application Modes lie in the common intersection, judiciously involving applications, the OS, and the user when necessary.

Starting from OS-only support, ACPI-related and CPU voltage- and frequency-scaling techniques, along with TailEnder [3] and CatNap [7], try to automatically determine a global machine behavior and estimate its best configuration in light of energy savings. In some specific occasions, they can be configured by users, although it is not always clear how these global settings will affect individual application functionality and energy-saving promises. Commercial OSes also have measures for reducing active demand, such as Windows Phone 7.5's Battery Saver Mode, which disables background processes and lowers the screen brightness when the battery charge drops below a certain level, or Apple iOS's disallowing of background processes as a global policy. As a research prototype, Llama [4] is an adaptive energy-management system for mobile devices that matches energy consumption with user behavior via probabilistic adaptation of system settings based on expected recharge time. Our work supplements Llama by integrating both user and applications in the adaptation process. In all these cases, because the OS lacks semantic knowledge of the application, the obtained savings are limited.

In the absence of OS and application support, users are forced to guess which settings, behaviors, or apps correlate with energy usage, having sometimes to understand specific implementation and hardware details. The OS can cooperate with the user by offering more visibility and obtaining hints about desired behavior, such as lifetime. We include here independent user-level services that monitor other running applications. Examples include Koala [21] and Carat [17]. Carat uses crowd-sourced usage data to suggest energy savings based on the device's state and past usage of similar devices, and presents the expected improvement in battery life if the user kills each of the currently running apps. Application Modes provide an interesting complement by increasing the granularity of possible user actions, as she can choose to change their functionality rather than just terminating them.

Some applications attempt to optimize their energy use without support from the OS or input from the user. Beyond development-time profiling (*e.g.*, [15]), some developers change the behavior of the app depending on the battery level, like the Dropbox example in §5. These applications may not have an obvious choice, though, without user input, when there are different changes in functionality in the Pareto frontier (*cf.* §2). Not surprisingly, some apps do involve the user. Sygic [23] is a voice-guided GPS navigation app that offers users distinct modes of operation based on different power-management profiles. While this approach is beneficial, not involving the OS has drawbacks. In the best case, there is a severe duplication of effort by independent developers, and in the worst case these apps will make suboptimal decisions, given the lack of coordination and the diversity of platforms on which they run.

Despite an initial effort to involve the user in energy-aware decisions [10], many recent solutions have focused on the energy-saving cooperation of the OS and app developers, perhaps to avoid alienating the user. A-States [12] are similar to Application Modes as they propose a narrow interface between apps and the OS to enable the coordination of power-saving measures without exchanging seman-

tic data. Our work differs in that we involve the end-user in the decisions, as the desirability of different modes, in our setting, is not monotonic, but can exhibit user-specific trade-offs. Eon [22] and Energy Levels [13] provide programming abstractions and runtimes to predict resource usage in wireless sensor networks (WSNs) and to automatically meet lifetime goals by deactivating or adapting parts of an application. Resource-usage prediction is facilitated by the single-threaded environment and periodic behavior of WSN applications. Users are likely the developers, who possibly understand the innards of such a complex system and there are almost no concerns about usability. Although Application Modes provide a similar abstraction and runtime, our focus is on multi-tasked, quasi-acyclic mobile applications that involves non-expert users.

Android's wakelocks also focus on this type of cooperation by allowing the kernel and user-space apps to control hardware suspending via reference counters to specific system components. To avoid races between suspend and wakeup events, all user-space processes need to be aware of the wakelocks interface. This is acceptable for Android, but not applicable to other Linux-based systems [26]. Developers have proposed alternatives to wakelocks for the mainline Linux kernel. Runtime Power Management[6] is similar to wakelocks, but is restricted to driver interaction with I/O devices. Autosleep[7] is a direct substitute which reuses components present in the mainline kernel. Applications could take advantage of both functionalities to cooperate with device drivers and provide hints on when to suspend.

There are very few works that, like ours, involve support from the three camps. Aside from Flinn and Satyanarayanan's energy-related extensions to Odyssey [10], Ghosts in the Machine [1] suggest giving the OS more visibility on the power state of I/O devices. Applications are adapted to issue hints to the device's power manager about their execution and help set the right power state, resulting in performance and energy-conservation improvements. Users express fidelity desires using a single unit-less knob that prioritizes performance or energy savings. While it is fundamental to limit the amount of input from the user, the diversity of mobile apps in our context makes this single dimension too restrictive to express user preferences, as different users will have preferences in the Pareto frontier that are not obvious to the developer or to the OS.

7. CONCLUSION

Application Modes enable the cooperation of the OS, applications, and users for efficient energy management in mobile devices. The user's only concerns are about differences in functionality and their impact on battery life. Application Modes are particularly well suited to cases in which there are multiple dimensions involved in the users' implicit preference function, with no total order among them. Applications provide the OS with discrete modes that express graceful degradation in face of limited energy. Further, the OS centralizes all of the knowledge and models about the current and future energy contexts, removing this burden from apps. We plan to use our abstraction and interface prototype for Android to instrument more applications, and conduct full-platform power measurements on real devices. The real measure of success for an interface is adoption, and we plan on conducting user studies with both developers and end-users.

Acknowledgments We thank Prabal Dutta, Srikanth Kandula, Deepak Ganesan, Shriram Krishnamurthi, Jonathan Mace, the anonymous reviewers, and our shepherd, Landon Cox, for their feedback. Marcelo was funded in part by a generous gift from Intel Corporation.

[6] http://lwn.net/Articles/347573

[7] http://lwn.net/Articles/479841

8. REFERENCES

[1] M. Anand, E. B. Nightingale, and J. Flinn. Ghosts in the machine: Interfaces for better power management. In *MobiSys'04*.

[2] R. K. Balan, M. Satyanarayanan, S. Y. Park, and T. Okoshi. Tactics-based remote execution for mobile computing. In *MobiSys '03*.

[3] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. Energy consumption in mobile phones: a measurement study and implications for network applications. In *IMC'09*.

[4] N. Banerjee, A. Rahmati, M. D. Corner, S. Rollins, and L. Zhong. Users and batteries: Interactions and adaptive energy management in mobile systems. In *Ubicomp'07*.

[5] L. A. Barroso and U. Hölzle. The case for energy-proportional computing. *Computer*, 40(12):33–37, 2007.

[6] D. Chu, A. Kansal, J. Liu, and F. Zhao. Mobile apps: It's time to move up to CondOS. In *HotOS'11*.

[7] F. R. Dogar, P. Steenkiste, and K. Papagiannaki. Catnap: Exploiting high bandwidth wireless interfaces to save energy for mobile devices. In *MobiSys'10*.

[8] C. S. Ellis. The case for higher-level power management. In *HotOS'99*.

[9] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan, and D. Estrin. Diversity in smartphone usage. In *MobiSys'10*.

[10] J. Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. In *SOSP'99*.

[11] W. Jung, C. Kang, C. Yoon, D. Kim, and H. Cha. Appscope: Application energy metering framework for Android smartphone using kernel activity monitoring. In *USENIX ATC'12*.

[12] A. Kansal, J. Liu, A. Singh, R. Nathuji, and T. Abdelzaher. Semantic-less coordination of power management and application performance. In *HotPower'09*.

[13] A. Lachenmann, P. J. Marrón, D. Minder, and K. Rothermel. Meeting lifetime goals with energy levels. In *SenSys'07*.

[14] X. Liu, P. Shenoy, and M. D. Corner. Chameleon: Application-level power management. *IEEE Trans. on Mob. Comp.*, 7(8), 2008.

[15] R. Mittal, A. Kansal, and R. Chandra. Empowering developers to estimate app energy consumption. In *MobiCom'12*.

[16] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, and K. R. Walker. Agile application-aware adaptation for mobility. In *SOSP '97*.

[17] A. J. Oliner, A. P. Iyer, E. Lagerspetz, S. Tarkoma, and I. Stoica. Carat: Collaborative energy debugging for mobile devices. In *HotDep'12*.

[18] A. Pathak, Y. C. Hu, and M. Zhang. Fine grained energy accounting on smartphones with eprof. In *EuroSys'12*.

[19] A. Roy, S. M. Rumble, R. Stutsman, P. Levis, D. Mazières, and N. Zeldovich. Energy management in mobile devices with the Cinder operating system. In *EuroSys'11*.

[20] M. Satyanarayanan and D. Narayanan. Multi-fidelity algorithms for interactive mobile applications. *Wirel. Netw.*, 7(6).

[21] D. C. Snowdon, E. Le Sueur, S. M. Petters, and G. Heiser. Koala: A platform for OS-level power management. In *EuroSys'09*.

[22] J. Sorber, A. Kostadinov, M. Garber, M. Brennan, M. D. Corner, and E. D. Berger. Eon: A language and runtime system for perpetual systems. In *Sensys'07*.

[23] Sygic. GPS navigation for smart phones. http://www.sygic.com.

[24] K. N. Truong, J. A. Kientz, T. Sohn, A. Rosezwieg, A. Fonville, and T. Smith. The design and evaluation of a task-centered battery interface. In *Ubicomp'10*.

[25] N. Vallina-Rodriguez, P. Hui, J. Crowcroft, and A. Rice. Exhausting battery statistics: Understanding the energy demands on mobile handsets. In *MobiHeld'10*.

[26] R. J. Wysocki. Technical background of the Android suspend blockers controversy. http://lwn.net/images/pdf/suspend_blockers.pdf.

[27] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat. ECOSystem: Managing energy as a first class operating system resource. In *ASPLOS-X'02*.

[28] 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'10*.

The Case for Psychological Computing

Xuan Bao, Duke University
Mahanth Gowda, Duke University
Ratul Mahajan, Microsoft Research
Romit Roy Choudhury, Duke University

ABSTRACT

This paper envisions a new research direction that we call *psychological computing*. The key observation is that, even though computing systems are missioned to satisfy human needs, there has been little attempt to bring understandings of human need/psychology into core system design. This paper makes the case that percolating psychological insights deeper into the computing layers is valuable, even essential. Through examples from content caching, vehicular systems, and network scheduling, we argue that psychological awareness can not only offer performance gains to known technological problems, but also spawn new kinds of systems that are difficult to conceive otherwise.

1. INTRODUCTION

Many fields such as economics, business and medicine have successfully leveraged psychological traits of humans to design better solutions [1–3]. In computing, researchers in human-computer interaction (HCI) have long recognized the need to embrace insights from human psychology [4–7]. This need is clear because the way users interface with technology depends on their psychological characteristics.

In this paper, we ask if computing systems can benefit from embedding human psychological factors deeper into their design and not just their interfaces. We begin by describing the origin of our inspiration—*RedBox*, a DVD rental company. We argue that psychological factors that help RedBox succeed can alleviate the crisis of bandwidth scarcity in cellular networks.

RedBox is a young company in the USA that sets up red-colored kiosks in accessible locations such as grocery stores, gas stations, and airports. The kiosks rent DVDs for a low price (often $1) [8]. People who pass by these kiosks, say when grocery shopping, check-out a DVD and return it the next day. RedBox has been tremendously successful at the same time that larger DVD rental stores (e.g., BlockBuster) are losing business. What is surprising is that with just 50 DVDs per kiosk, and a few kiosks per zip code, RedBox is able to cater to the needs of the population. People are not driving to large rental stores with a much wider selection of DVDs; many are happy picking a DVD from the limited options that a RedBox kiosk offers.

We posit that RedBox's success can be explained by psychological traits of humans. Human desires, especially those pertaining to leisure and entertainment, are not always rigid and exhibit flexibility [9]. A customer may be in the mood for an action movie, but may not have a specific movie in mind. This customer's needs can be met as long as the kiosk has at least one movie that falls within her "circle of flexibility." Further, human flexibility can be increased through incentives, and their choices can be influenced by stimuli in the environment. The incentive for using a RedBox kiosk is its cheaper price and easier accessibility than a larger store. These factors may influence a customer to even rent a comedy if none of the available action movies are to her liking.

We explore the possibility of leveraging the same psychological factors to alleviate the bandwidth crisis in cellular networks. Cellular data networks are facing a serious crisis in coping with increasing demands, especially of video content, from mobile devices [10]. One promising solution is prefetching, i.e., predicting the content that the user is going to request in the future and downloading them over WiFi. Unfortunately, such prefetching has proved extremely difficult; one almost needs a crystal ball to accurately predict the content that the user will request a few hours in the future, time scales at which WiFi-based prefetching can help. Hence, most schemes prefetch at much shorter time scales, e.g., top search results can be prefetched if the user searches for a keyword in Google. While this improves user experience, it does not reduce 3G load [11].

Now consider how psychological factors above can change the situation. Alice's mobile device can roughly speculate about her flexible content consumption interests (perhaps based on her Web browsing history), and prefetch related videos over WiFi. The prefetched videos need not precisely match what Alice would later request; a reasonable set will suffice. Now, when Alice wants to view videos and only 3G connectivity is available, she is shown the list of prefetched videos. Of course, she can stream a non-cached video over 3G (equivalent to driving to BlockBuster), but because of flexibility and incentives, she may often be happy with a prefetched video (RedBox kiosk). The incentive is that prefetched videos play uninterrupted–streaming over 3G can be jittery—and do not contribute to her quota of cellular data. Our preliminary experiments indicate that, with such a prefetching scheme, on average users view 2 out of 3 videos from the cache. This indicates a promising opportunity to reduce network load.

In the body of the paper, we argue that the application of psychology is broadly useful and provide several examples of such systems. Besides improving existing techniques such as prefetching, psychological insights can facilitate new kinds of systems. Consider driving, for instance. It is well known that drivers are often distracted by emails and SMSs. Consequently, governments are banning their use [12], and researchers are developing technical solutions to detect when the user is driving and disable the phone [13]. However, recent psychological studies show that in certain conditions

HotMobile'13, February 26–27, 2013, Jekyll Island, Georgia, USA.

driving without any distraction incurs an excessively low cognitive load [14], inducing drowsiness and day-dreaming. This implies that too little mental load is risky, just like too much mental load. Taking this psychological factor into account, we propose that smartphones should detect when the driver is dozing off or drifting in thought, and intelligently present "information snacks" to maintain the driver's cognitive load above a threshold. Such snacks could be in the form of read-aloud emails, a quick display of a Facebook picture, or games that require the driver to focus on the road. Our preliminary experiments demonstrate the effectiveness of this approach. Conceiving such systems requires co-thinking psychology and technology, rather than studying them in isolation.

Current technological trends are ripe to begin thinking in this direction. As we move from personal *computers* to personal *computing*, the need for devices to intimately understand users is greater. Fortunately, the convergence of smartphone sensors (always on humans), smart objects in the surroundings (measuring context), and recent advancements in machine learning (making decisions from big data), make it feasible to gain deep insights into human behavior and mood. Quantitative psychology, a burgeoning field, is already taking advantage of this convergence. The advantages could be mutually reinforcing—technology could gain from psychological insights that were in turn gained from technology.

This paper barely scratches the surface of what psychological computing could be and puts forth assorted ideas to motivate the case. We are aware that we are at a rather early stage. Nonetheless, we believe that there is latent promise, and we invite the community to give us feedback and get involved. In collaboration with psychologists, important technological changes are feasible. At the least, we believe it is worth trying.

2. MOTIVATING EXAMPLES

We motivate the need for factoring in psychological traits in system design by outlining several diverse examples. For each example, we present the psychological traits being exploited and explain how they facilitate system designs that break away from convention. For two of the examples, we also present preliminary experimental results.

2.1 Overnight Pre-fetching and Caching

(Content Demand is Flexible)

Our first example is that of a system that prefetches content when bandwidth is cheap (e.g., WiFi) for later consumption when it is expensive or unreliable (e.g., 3G). It is based on the following three observations about psychological behavior, which we believe also underlie the success of RedBox as a content dissemination platform.

(1) **Human demands for some types of content (e.g., videos) are flexible.** Human demands for content are not always rigid. Often, people do not have a specific content in mind, but are open to suggestions. This openness is part of the reason that people can find watchable movies from a RedBox kiosk with only 50 DVDs.

(2) **Human flexibility can be magnified with incentives [15].** Some customers may prefer other movies than those in the kiosk but still pick from the kiosk rather than going to other places. Two incentives contribute to this behavior. The first is accessibility: the kiosks are distributed at convenient places where people naturally visit. The second is price: Redbox movie costs $1, while bigger stores cost around $4.

(3) **Fewer choices simplify decision making.** While one may think of Redbox's limited movie choices as a disadvantage, psychological studies show that fewer choices can actually be advantageous because it eases the decision-making process [16]. When too many choices are present, humans may make no choice at all because of the excessive cognitive overhead in making the right decision.

The psychological factors above suggest that a content prefetching system can succeed by caching even a small library if users have flexibility in content consumption. To validate this hypothesis, we build a system to save 3G bandwidth. The mobile device, instead of downloading contents through 3G on the fly, can prefetch videos when WiFi is available, e.g., overnight.

2.1.1 Prototype Design

While the observations above and our techniques apply to many forms of content, our prototype focuses on YouTube videos because video content consumes a large amount network bandwidth today and its share is expected to further grow. Designing our system entails two main challenges. First, we must infer the user's interests at a coarse level; no incentivization may convince a user to consume content in which they are not even remotely interested. Second, we must identify a small set of videos to prefetch such that the user is likely to consume a video from this set.

An overview of our system is shown in Figure 1. The interest mining module infers the user's interest by mining their personal information such as Web history, email content, and social network feed. Given historical data of these activities, we use an NLP technique called SIP (Statistically Improbable Phrase Extraction) [17] to mine for interests. SIP essentially finds meaningful phrase combinations in textual data that summarize the text. Table 1 shows a subset of interests for three of our study participants.

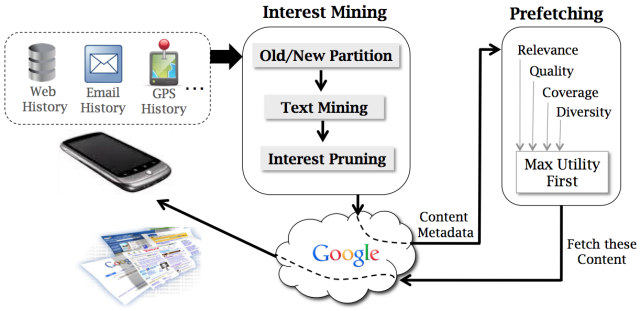


Figure 1: Overview of overnight caching

Table 1: Example Interests

User1(Wireless)	User2(Game)	User3(Mobile)
Distributed Comp.	New Vegas	Efficient Retrieval
Summer Intern.	Civilization V	Life Log Based
Software Radio	Scroll V	Anatomy Netflix
Distributed Sys.	The Vault	

We then use the interests to identify a set of matching videos through YouTube search queries. The *Prefetching* module de-

cides which videos to prefetch. It considers not only video relevance and popularity but also the degree of diversity among cached videos and the level of coverage they provide for the topics of interest. For example, if the user exhibits interests in both soccer and volleyball, we cache at least a few volleyball videos as well, even though soccer videos tend to be much more popular. Further, the relationship between videos is also considered. Since YouTube users often follow the "related video" links after watching a video, the module chooses to cache some of these related videos as well [18]. Our goal is to maximize the possibility that the user will find at least some cached content that falls within her circle of flexibility.

The prefetched content is advertised in a side-panel whenever the user visits YouTube. This serves the role of explicit incentivization towards cached videos. The list length is limited to 50 for a manageable browsing and selection effort.

2.1.2 Preliminary Results

We conducted a preliminary user study to evaluate our hypothesis and techniques. For each user, we first identified their interests based on their browsing history, and for each inferred interest, we asked them if it was relevant. We then prefetched videos based on those interests, and for each video, we asked them if it was relevant (i.e., they would like to view it at some point in time). Finally, we asked them to browse YouTube for a fixed duration, during which they were free to view prefetched or non-prefetched videos or a mix. We recruited eight subjects, of which one subject only finished the interest relevance portion of the study.

Figure 2 shows the performance of interest mining by plotting the percentage of the mined topics that the subjects deemed relevant. The average across users is 66%, which means that even simple interest mining techniques can identify a user's interests.

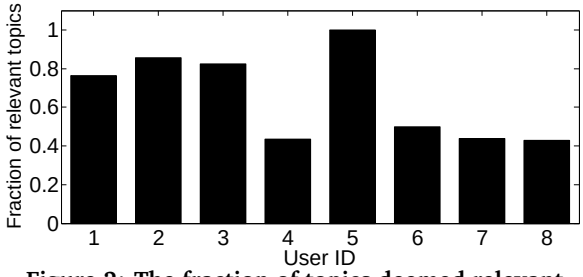


Figure 2: The fraction of topics deemed relevant.

Figure 3 shows the fraction of prefetched videos that the subjects deemed relevant. We see that this fraction varies from 10-60%, with the average being 40%. We find this encouraging given that the YouTube library is vast, in which only a minuscule fraction is presumably relevant to a given user. We show below that, because of psychological factors mentioned above, this relevance rate is enough for users to consume a significant fraction of the content from the cache. More sophisticated techniques (e.g., those based on collaborative filtering) may further boost the relevance rate.

Figure 4 shows the comparison between cached and on-the-fly videos users watched during the study. We find that 50-75% of the videos viewed by the subjects were prefetched, with the average being 65%. That is, two out of three viewed videos were from the cache. These preliminary result suggests

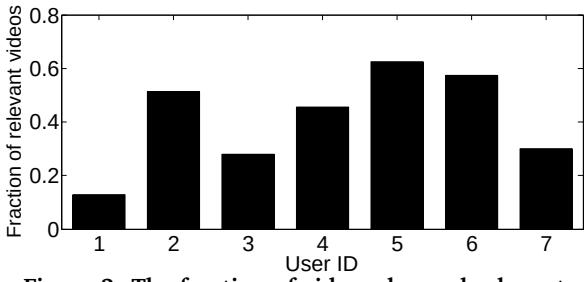


Figure 3: The fraction of videos deemed relevant.

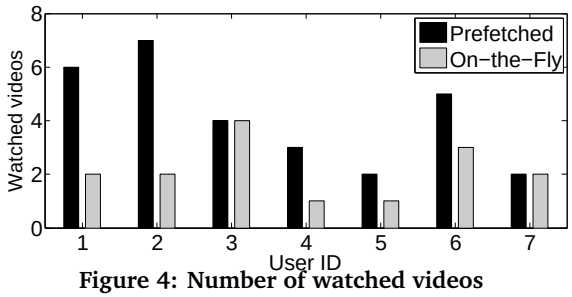


Figure 4: Number of watched videos

that our techniques can lead to a significant reduction in 3G bandwidth consumed by users. Our experiments did not involve economic incentive towards cached videos—there was only a performance incentive. We expect higher cache usage when economic incentives are involved.

2.2 Driving with Cognitive Comfort

(Controlled Diversion is Beneficial)

Driving in the USA can sometimes be monotonous. Highways can be empty, the view can be dull, and adaptive cruise-control and blind-spot detectors can eliminate the sudden surprise. In conjunction, the complete ban on phone-use while driving is eliminating the periodic diversions. Such driving conditions can translate to an excessively low cognitive load on the driver, perhaps resulting in the driver dozing off or drifting away in thought. Importantly, a DoT report and related research papers already show evidences of such cases – called highway hypnosis [14]. The report predicts that the problem is likely to become pronounced over time as cars become semi-autonomous, leaving the driver with very little work on average.

We view an opportunity for psychological computing here. Our hypothesis is that a controlled level of diversion may be beneficial to keep the cognitive load on the driver above a threshold, which can help with remaining focused on driving. To this end, we propose a system in which the driver's smartphone is mounted on the car's dashboard. The phone looks for indications of dozing or hypnosis (such as through closing eyelids and head stillness), and on detecting them, presents the driver with electronic diversions. Such diversions could be emails read out aloud, a picture displayed from a social network feed, or even a game that forces the driver to focus on the road. We call these diversions "information snacks" to capture how they feed the driver's mind and keep her vigilant about road conditions.

We designed a user study to examine the merit of controlled diversions during driving. The study is inspired by the standard psychological experiment – called *MackWorth Clock* – that tests for human vigilance [19]. In MackWorth Clock test, users are required to focus on the seconds hand of a clock,

and press a button whenever the hand skips a tick. The skips are infrequent, measuring the extent to which the user can remain vigilant under such monotonicity. We adapted the MacWorth clock experiment, but replaced the clock with a 30 minute video recorded from a car's windshield. The video is monotonous because it is mostly recorded on an empty US highway, however, at infrequent time points, a few frames of the video are removed, causing a distinct glitch. We created a web-based interface and asked volunteers to press a key whenever they noticed the glitch. During some part of the video, we periodically presented information snacks in the form of audio and visual diversions. The diversions were read-aloud news headlines and interesting facts, or interesting pictures chosen from Flickr. A screenshot of our experiment interface is shown in Figure 5(a), with a picture of a deer as a visual diversion. We measured the accuracy of glitch detection with, and without, diversions.

We find that without diversion, the accuracy proved to be 64.2%, but when presented with diversions, users achieved 83.3% accuracy. We also noticed improved detection when the diversions were presented tens of seconds before the glitch, perhaps suggesting that diversions increase human alertness, which then decays. If such behavior holds in real-life driving, we conjecture that information snacks could be effective there as well.

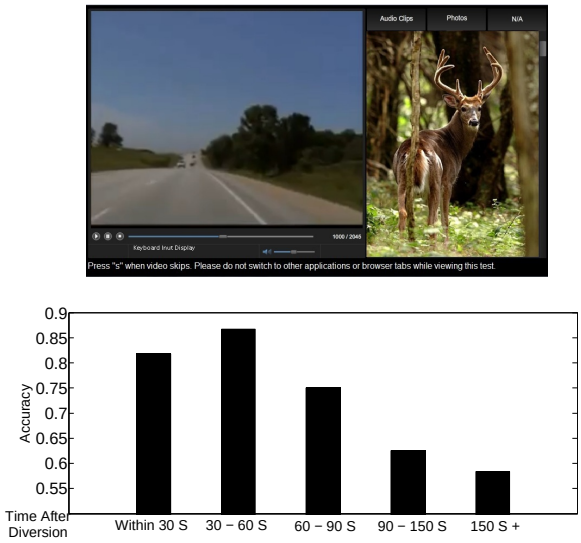


Figure 5: (a) Screenshot from the driving diversion study. (b) Glitch detection accuracy higher just after the diversion, but decays over time.

2.3 Cognitive Load Aware OS

(Will Power is Like Muscle)

Many tasks on a computing device (e.g., coding, writing) incur high cognitive load. Recent research shows that the main resource exerted for such tasks—will power—is akin to a muscle [20]. It gets tired and becomes less effective as it is used.

Based on this finding, we propose that computing devices should help humans work more effectively. These devices are in a good position to judge the depletion level of the user's will power, derived from the nature of the task, the length of time she is working, and her rate of errors (e.g., number of typos). When the system observes a well-focused user, it could perhaps delay her incoming emails and other forms of electronic distraction, even place her phone on least-activity mode. When the focus begins to deplete, emails and social feeds could be scheduled to arrive, and the user could even be prompted to take a break.

2.4 Eagerness Aware Scheduling

(Perception of Time is Relative)

Human perception of "elapsed time" is inaccurate and often a function of the situation. Attending a boring class for five minutes may appear far longer than watching a gripping video for half an hour. This suggests that any form of system delay, as experienced by humans, may not be measured in sheer time difference between a request and response. Rather, the user's experience could be a function of the "perceived time". If the network could receive hints about the user's current perception (say, the user is waiting alone at a bus stop), her transmission could be prioritized to reduce her perceived waiting time. On the other hand, a student chatting on IM may start browsing videos at the same time – such a student need not be prioritized when the system resources are constrained. To generalize this, networking systems could account for human eagerness while scheduling resources. This is of course different from traditional techniques that prioritize purely based on the application category, i.e., real-time applications over delay-tolerant ones.

2.5 Psychology Aware Compression

(Unequal Focus on Pixels)

Conventional image compression techniques utilize understandings of human visual properties. However, these schemes often do not consider the semantic meaning of the image content and its implication on human perception. While viewing a content, say an image or a video, a user attaches unequal importance to different parts of the content. Psychological studies have shown that when the content theme is human relationships, for instance, viewers' eye movements are biased to human faces, to the extent that one can miss the presence of other objects in the scene [21]. Figure 6 is extracted from the famous Yardbus experiment in 1967, showing an example of a picture and the corresponding eye movements. If such disparities exist in the way viewers "consume" pixels, compression schemes can take advantage of them. Depending on the theme, certain parts of the content can be retained at high fidelity, and others treated with a higher compression factor.

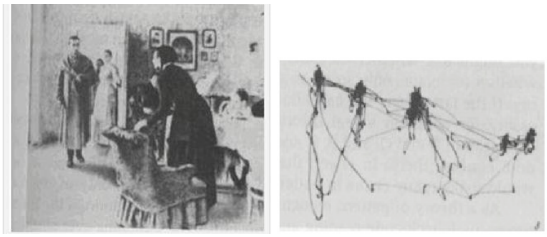


Figure 6: YardBus' study showing how viewers' eye movements are biased towards faces when human relationships are the subject of the picture.

In bandwidth or latency constrained operations, such tradeoffs may be welcome. Graphics rendering could also exploit the above observation. Some parts of the images can be rendered at lower fidelity, saving computational cost and energy.

3. FRAMEWORK AND DISCUSSION

This vision paper is an early exposition of formative ideas and intuitions. We discuss some of the open questions here.

3.1 PSY Layer Implementation

The notion of human psychology is obviously complex and context-sensitive. A key challenge is to make suitable abstractions that can be systematically used in computing systems.

Our intuition is that psychological attributes may be viewed as operating at two time-scales, and should be treated differently. The first class is composed of slowly-changing or stable attributes, those that correspond to "inherent nature" or habits (e.g., flexibility in video demand). The system can directly apply this knowledge (e.g., an empirical flexibility factor) for prefetching and caching. The second class is composed of short time-scale, "state-dependent" attributes, those that impact users' behavior, actions, and current mental state (e.g., cognitive load). Of course, the system would need to capture these "state" information proactively via sensing.

Our deliberations on an architectural framework for psychological computing resulted in more than one design choice. Figure 7 shows one possible framework. In this design, we envision a psychological module (PSY) embedded into the OS of the mobile device. This module implements common services related to psychological computing from which many applications can benefit, to minimize duplication of effort across applications. Embedding PSY in the OS also makes the OS aware of psychology-related factors, and PSY benefits from the context-inferencing functionalities in the OS [22].

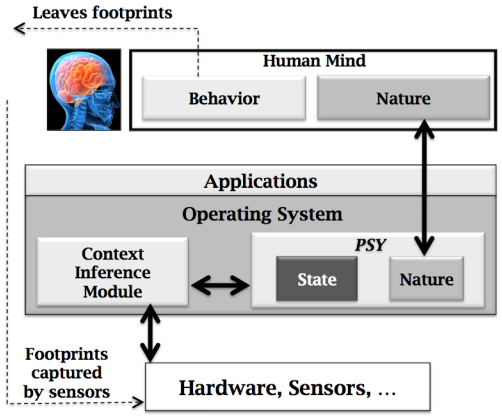


Figure 7: A framework for psychological computing.

The proposed PSY module consists of two components. The "Nature" module is modeled after the "inherent nature" of humans. It can be viewed as a storage of fixed parameters and predefined logic that are consistent with common/expected behavior. To capture these, the "Nature" module in PSY needs to assimilate knowledge about the user (e.g., how flexible a user is), and more importantly, incorporate new findings from modern psychology and brain sciences. The "State" module offers a library of current psychological characteristics and behaviors, through both request-response APIs and publish-subscribe APIs. Example APIs include *currentCognitiveLoad()*, *impatienceIndex()*, *mood()*, *onCognitiveLoadChange()*, and when possible, some physiological indicators, such as *heartRate()* and *pulseOximeter()*. The OS, network stack, and applications draw from both the long and short term classes, depending on their goal. Table 2 shows how a few example applications can be supported by this framework.

Table 2: Support Example Applications

Example Applications	Module	Variable
Prefetching	Nature	Flexibility
Compression	Nature	Visual Attention Bias
OS, Driving, Scheduling	State	Cognitive Load

As stated before, this architecture is not the only possible design. Another alternatives is a cross-layer design that incorporates different levels of behavioral information separately (e.g., facial expressions (smile, frown, etc) could be an example of explicit, low-level information. On the other hand, the user's mood may be a sophisticated, high-level state.). We converged on the Nature-State archictecture because we found it most useful towards implementing the examples we presented. But we do realize that we are early in our thinking and significantly more experimental work is needed to produce a good design.

3.2 Challenges

Realizing the framework above entails many challenges. We discuss two of the main challenges here.

The first challenge relates to identifying the psychological traits that can be exploited towards system design. Admittedly, psychology is a deep field by itself. Limited by our current understanding, most of the discussion in this paper relates only to behavioral and perceptual psychology, especially the quantitatively measurable portion. Fascinating studies in these fields have uncovered many human biases and traits. In addition to the traits mentioned previously, some of them are: (1) humans simplify their decisions based on implicit comparisons, e.g., when given a choice between a paid vacation to Rome, a paid vacation to Paris, and paid vacation to Paris except for the coffee charges, a large fraction of users chose the second option. This is because while comparing between Paris and Rome is difficult, the benefit of free coffee is obvious. Therefore, the second option suddenly looks like a bargain compared to the other two choices. (2) Fatigue can be alleviated if a human switches to a different task that stimulates a different part of the brain. We wonder if these, and various other psychological properties of the mind, can be accommodated in systems of the future.

The second challenge relates to inferring the users' current state and inherent nature. The field of behavioral psychology focuses on observable behavior, and their link to psychological conditions. Advancements in sensing and pattern recognition technology is helping this field uncover fascinating connections. For instance, a recent study shows that the cognitive load on a person can be measured based on pupil dilation [23]. Recent works by Picard et. al. from MIT demonstrates the ability to recognize a user's heart rate by observing her through a webcam [24]. Progress in psychological computing will rely heavily on such advancements and cross-disciplinary research will be vital.

3.3 Natural Questions

(1) **Psychological computing may often leverage the user's context. In that sense, is this essentially context-aware computing?** Context is a broad term defined by the "circumstances that form the setting of an event or object" [25].

While a large body of past work has leveraged the physical context (e.g., ambience, location, time, activity), we are unaware of any systematic effort to exploit the psychological context in core technology. Of course, the psychological context may sometimes manifest itself through physical contexts, such as a person dozing off because of a low cognitive load. We aim to measure the physical activity of dozing off, but utilize the psychological context underneath (i.e., low cognitive load) for system design. It is the use of this hidden variable, we believe, that differentiates psychological computing from context-aware systems.

(2) **Why is psychological computing not a pure HCI problem?** HCI is obviously positioned to leverage psychological factors. However, we believe that incorporating psychological primitives into the "system core" may offer significant benefits beyond a pleasant user experience. For example, many applications, including comfortable driving and eagerness-aware scheduling, can be supported by a common *cognitive load API* exposed by the OS. Perhaps more importantly, the benefits here do not relate to only the users' experience; the system itself benefits by making more efficient resource allocation (CPU cycles, battery, storage).

(3) **How to prevent the system from misbehaving seriously when it misinterprets users' mental states?** Like context-aware systems, the accuracy of psychology-aware systems may not be perfect. Therefore, how to prevent the system from annoying the user when such errors occur is an important challenge. While we do not have a full solution, erring on the side of conservativeness may be a helpful guideline. For instance, in the cognitive load-aware OS example, the system should gradually increase the level of task shuffling. It should trigger a psychology-aware action only after it has gained adequate confidence in its inferences. Moreover, if the user ever manually overrides the system's decision, the system should back-off and not disturb the user until the user specifies otherwise. In other words, the system should avoid being intrusive and provide a scheme to fall back to a conventional system.

4. RELATED WORK

Outside the area of HCI [5–7], a few prior works have also exploited psychological theories for different purposes. For example, Verendel [26] investigates security issues taking rationality into consideration. Also, Shye et al. [27] leverage human physiological traits to control microprocessor frequency in order to save energy. The TUBE project [28] brings humans into the decision loop to alleviate network congestion. Building on these efforts, we argue for systematically embedding human psychology deeper into computing systems.

5. CONCLUSION

Computing systems were envisaged to serve human needs. We argue that, in our eagerness to build sophisticated systems, we may not have adequately comprehended the nature of these needs. Yet, the success metric of computing systems seem strongly influenced by these needs, and is likely to become more so, as technology percolates into every corner of our lives. This paper postulates that improved comprehension of human psychology can help improve the systems we build, both in terms of performance and relevance. We sketch early examples in cellular networking, vehicular systems, scheduling, and content encoding, and generalize them to a broader direction that we refer to as *psychological computing*. Although premature in its current state, we believe this direction has promise, and present it to the community for feedback, opinion, and involvement.

6. REFERENCES

[1] D. Kahneman. Maps of bounded rationality: Psychology for behavioral economics. *The American economic review*, 2003.
[2] G.R. Foxall et al. *Consumer psychology for marketing*.
[3] E.J. Khantzian. The self-medication hypothesis of substance use disorders: A reconsideration and recent applications. *Harvard Review of Psychiatry*, 1997.
[4] J.M. Carroll. *Designing interaction: Psychology at the human-computer interface*. Cambridge Univ Pr, 1991.
[5] S.K. Card, T.P. Moran, and A. Newell. *The psychology of human-computer interaction*. CRC, 1986.
[6] J. Fogarty, S.E. Hudson, and Others. Predicting human interruptibility with sensors. *ACM TOCHI*, 2005.
[7] C. Harrison, B. Amento, S. Kuznetsov, and R. Bell. Rethinking the progress bar. In *ACM UIST*.
[8] How redbox is changing retail. *Marketing News*, 2009.
[9] D. Ariely, G. Loewenstein, and D. Prelec. Coherent arbitrariness: Stable demand curves without stable preferences. *The Quarterly Journal of Economics*, 2003.
[10] S. Deb, K. Nagaraj, and V. Srinivasan. Mota: engineering an operator agnostic mobile service. In *ACM MobiCom*, 2011.
[11] D. Fisher and G. Saksena. Link prefetching in mozilla: A server-driven approach. *WCW*, 2004.
[12] D. Lamble et al. Mobile phone use while driving: public opinions on restrictions. *Transportation*, 2002.
[13] J. Yang, S. Sidhom, et al. Detecting driver phone use leveraging car speakers. In *ACM MobiCom*, 2011.
[14] L. Harms. Variation in drivers' cognitive load. effects of driving through village areas and rural junctions. *Ergonomics*, 1991.
[15] D. Ariely. *Predictably irrational: The hidden forces that shape our decisions*. HarperCollins, 2009.
[16] D. Kuksov and J.M. Villas-Boas. When more alternatives lead to less choice. *Marketing Science*, 2010.
[17] M. Errami et al. Identifying duplicate content using statistically improbable phrases. *Bioinformatics*, 2010.
[18] S. Khemmarat, R. Zhou, L. Gao, and M. Zink. Watching user generated videos with prefetching. In *ACM MMSyss*, 2011.
[19] NH Mackworth and et al. *Researches on the measurement of human performance*. 1950.
[20] D. Kahneman. *Thinking, Fast and Slow*. 2011.
[21] A.L. IArbus and A.L. IArbus. *Eye movements and vision*. Plenum press, 1967.
[22] D. Chu, A. Kansal, J. Liu, and F. Zhao. Mobile apps: It is time to move up to condos. In *HotOS*, 2011.
[23] F. Paas and et al. Cognitive load measurement as a means to advance cognitive load theory. *Educational psychologist*, 2003.
[24] R.W. Picard. Emotion research by the people, for the people. *Emotion Review*, 2010.
[25] A.K. Dey. Understanding and using context. *Personal and ubiquitous computing*, 2001.
[26] V. Verendel. A prospect theory approach to security. Technical report, Technical Report, 2008.
[27] A. Shye and Others. Power to the people: Leveraging human physiological traits to control microprocessor frequency. In *IEEE/ACM International Symposium on Microarchitecture*, 2008.
[28] S. Ha and Others. Tube: time-dependent pricing for mobile data. In *ACM SIGCOMM*, 2012.

InSight: Recognizing Humans without Face Recognition

He Wang
Duke University

Xuan Bao
Duke University

Romit Roy Choudhury
Duke University

Srihari Nelakuditi
University of South Carolina

ABSTRACT

Wearable cameras and displays, such as the Google Glass, are around the corner. This paper explores techniques that jointly leverage camera-enabled glasses and smartphones to recognize individuals in the visual surrounding. While face recognition would be one approach to this problem, we believe that it may not be always possible to see a person's face. Our technique is complementary to face recognition, and exploits the intuition that colors of clothes, decorations, and even human motion patterns, can together make up a "fingerprint". When leveraged systematically, it may be feasible to recognize individuals with reasonable consistency. This paper reports on our attempts, with early results from a prototype built on Android Galaxy phones and PivotHead's camera-enabled glasses. We call our system *InSight*.

Categories and Subject Descriptors

H.3.4 [**Information Storage and Retrieval**]: Systems and Software; C.2.4 [**Computer-Comunication Networks**]: Distributed Systems

General Terms

Design, Experimentation, Performance

Keywords

Wearable camera, visual fingerprinting, smartphones, augmented reality, matching, recursion, distributed cameras

1. INTRODUCTION

Imagine a near future where humans are carrying smartphones and wearing camera-embedded glasses, such as the Google Glass. This paper intends to recognize a human by looking at him or her from any angle, even when her face is not visible. For instance, Alice may look at people around her in a social gathering and see the names of each individual – like a virtual badge – suitably overlaid on her Google Glass display. Where revealing names is undesirable, only a tweet message could be shared. People at the airport could tweet "looking to share a cab ", and Alice could view each individual's tweet above their heads. In general, we intend to extend augmented reality [1, 2] to humans and the key challenge pertains to differentiating individuals. We explore options outside face recognition [3, 4].

Our core technique exploits the intuition that faces are not necessarily the only "visual fingerprint" of an individual. Features combined from clothing colors, body structure, and motion patterns can potentially be fingerprints for many practical scenarios. There is evidence of this opportunity given that humans can often recognize another human without looking at her face. This paper asks: *can sensor-enabled smartphones and wearable glasses together achieve the same?*

Our main idea is simple and illustrated through Figure 1. Whenever a user Bob uses his phone (such as while checking emails), the phone's camera opportunistically "takes a peek" at Bob. Through image segmentation and processing [5, 6] the phone extracts a visual fingerprint – a feature vector that includes clothing color and their spatial organization. The spatial organization captures the relative locations of each color in 2D space, hence a red over blue shirt is different from blue over red. This spatio-chromatic information – called Bob's *self-fingerprint* – is announced in the vicinity. Nearby smartphones receive a tuple:

$$S_{Bob} = \text{<Bob, Bob's self-fingerprint>}.$$

Now consider Alice (wearing a Google Glass and carrying a smartphone) looking at a group of people that includes Bob. A picture from the glass is processed on Alice's phone (or in the cloud), and through image segmentation and analysis, the phone computes each individual's spatio-chromatic fingerprint, F_i. Since Alice has separately received Bob's self-fingerprint, S_{Bob}, a matching algorithm computes the similarity between F_i and S_{Bob}. If one of the fingerprints, F_j matches strongly with S_{Bob}, then Alice's phone can recognize Bob against the group of people. An arrow labeled "Bob" can now be overlaid on the image segment that generated F_j; Alice can view this either on her Google Glass display or on her smartphone screen (as shown in Figure 2).

Realizing the above idea presents a number of challenges. Bob's self-fingerprint is likely to capture only some parts of his clothing, and may not be adequately discriminating (particularly when Alice views Bob from the back, or when Bob is partially visible in the crowd). Even if front and back fingerprints are somehow available, and Bob is fully visible, ambiguity can arise when people are wearing similar dresses, say in a birthday party

ACM HotMobile'13, February 26–27, 2013, Jekyll Island, Georgia, USA.

Figure 1: Sketch of *InSight*: Bob's phone announces his own name and fingerprint to the vicinity; Alice's phone computes fingerprints from her glass, matches them against the ones received from the vicinity, and recognizes Bob.

Figure 2: An arrow labeled "Bob" overlaid on Bob in the smartphone's screen.

with a dress theme. Finally, even when all is well, different lighting conditions, wrinkles on clothes, and human mobility can inject errors into the system. Coping with these challenges are indeed non-trivial, however, we believe that certain opportunities can help, as described next.

(1) Even if Bob's self-fingerprint is not highly discriminating, Charlie may luckily identify Bob when Bob happens to be alone or around a few people wearing sharply contrasting clothes. Since Charlie now sees Bob's full clothing, he could enrich Bob's self-fingerprint with more information and upload it to the cloud. The enriched self-fingerprint helps others discriminate Bob better, which in turn enables more frequent opportunities to enrich his fingerprint – *a recursive process*. We find that the system converges in our limited set of experiments, enriching almost everyone's self-fingerprints.

(2) If the system still contains visual ambiguity, we observe that short term motion can be exploited. When Bob moves at a certain pace in a certain direction, Bob's accelerometer and compass could compute his motion vector and include it in his self-fingerprint. Alice could compute a similar motion vector from a short Google Glass video (by comparing a few consecutive video frames [7]), and match against all the motion vectors embedded in received self-fingerprints. If the best match corresponds to Bob, then Alice can disambiguate Bob despite visual similarity.

This paper is targeted to harness these opportunities. Our early prototype, built on Android phones and PivotHead camera-equipped glasses, implements basic self fingerprinting and matching. Offline experiments with 15 people in a clique yield promising results – 93% of correct recognition when viewed from the front. When viewed from the back, the accuracy degrades sharply. However, when different views are used to recursively enrich fingerprints (implemented via *monte carlo* simulations), the system converges to 96% accuracy even when viewed from the back; the front-side accuracy is perfect, and the convergence time is not long. Overall, *InSight* is an autonomous, self-correcting scheme, much different from a crude color matching idea proposed in our earlier work [8]. If successful, *InSight* could perhaps trigger new thinking in human-centric augmented reality applications [9].

2. SYSTEM SETTING

InSight assumes that Bob uses his smartphone to check emails or browse the Internet. When the phone is held in a specific orientation and the display senses finger taps – partly ensuring that the front-facing camera is facing Bob's upper body – the phone takes a few opportunistic pictures[1]. The pictures are analyzed, visual fingerprints extracted, and concatenated with the name "Bob", or any content/tweets that Bob intends to make visible. This self-fingerprint is either announced to the vicinity via Bluetooth Low Energy (BLE) or transmitted to the cloud along with the Bob's rough location. With BLE beacons, nearby phones directly receive the fingerprint. For cloud-based access, all phones update the cloud with their locations; the cloud matches the fingerprints and pushes back the recognition results. While both approaches present tradeoffs, we use the cloud based approach. As we will see later, the cloud based approach allows a central repository of fingerprints that can be distributedly updated by different people over time.

A viewer Alice looks at different people, and when in need to recognize a person, presses a button on her camera-enabled glass. A short video – of around 3 seconds – is recorded and transferred to her smartphone via WiFi or BLE, whichever is available on the glass. In the default case, the phone processes one of the frames in this video, separates different individuals in this image, and extracts visual fingerprints corresponding to each of them. For each computed fingerprint (sent to the cloud), the cloud computes a "similarity" with Bob's self-fingerprint. When the similarity is greater than a high confidence threshold,

[1]We discuss privacy issues in Section 5.

the cloud identifies Bob – Alice's phone superimposes an arrow on her phone screen or the glass display. When the similarity is sub-threshold, InSight explores motion patterns (speed and walking direction) for better recognition.

A natural question might be: *why not utilize the user's location as a form of identification?* While this is indeed a possible approach, we believe that such precise location in indoor spaces is unavailable today. Moreover, it may not be easy to extract depth information from the video, i.e., if Alice is looking down a corridor, its unclear what "location" she is looking at. Finally, locations need to be combined with compasses to compute the line of sight of the viewer; given compasses have a reasonably large error, especially indoors, pure location-based solutions may not suffice. However, location and compasses can be used to narrow down the search space for visual recognition. While we don't leverage this opportunity (to understand the limits of our techniques), we certainly intend to optimize InSight with location, compass, and face recognition in the future.

3. SYSTEM DESIGN

We sketch the basic design decisions underlying *InSight.* Several deliberations underpinning these decisions are curtailed in the interest of space.

3.1 Extracting Self-Fingerprints

Figure 3(a) shows an example photo taken opportunistically by InSight[2]. The key task here is to extract a visual self-fingerprint that is robust to lighting conditions, viewing angle, and viewing distance. Put differently, even if different people look at the same person from different positions (Figure 3(b)), the fingerprint from all these views should reasonably match the self-fingerprint. As a first step, InSight automatically crops out a rectangular region from picture – the part below the face/neck. It then applies two well known techniques on the cropped image, namely (1) spatiograms, and (2) wavelets.

Figure 3: (a) Upper body view when user browsing on his smartphone (b) View from user wearing a glass.

(1) Spatiograms: Spatiograms are essentially color histograms with spatial distributions encoded in its structure. Put differently, while basic color histograms only capture the relative frequency of each color, spatiograms capture how these colors are distributed in 2D space. The second order of spatiogram can be represented as [10]:

$$h_I(b) =< n_b, \mu_b, \sigma_b >, \quad b = 1,2,3,\cdots,B,$$

where B is the number of color bins, n_b is the number of pixels whose value falls in the b^{th} bin, and μ_b and σ_b are the mean vector and covariance matrices of the coordinates of those pixels respectively. Through such a representation, a white over red stripe can be distinguished from a red over white stripe, even if the number of red and white pixels are identical in both. Also, to cope with various viewing distances, we normalize the spatial information with respect to the *shoulder width* so that all the spatial representation is relative to the captured body size in each photo. Finally, to decouple lighting conditions from the colors, we convert the pixels from RGB to HSV, and quantize them into $B = 10x4x4$ bins.

(2) Wavelets: Apparels are often fashioned with patterns that run horizontally, vertically, or along diagonals. InSight captures them by computing the energy distribution over wavelet sub-bands [11, 12] along the vertical (E_v), horizontal (E_h) and diagonal (E_d) dimensions. As a result, different organizations of edges exhibit distinct feature vectors in our representation (Figure 4). We also use the ratio between E_v and E_h to improve robustness against different viewing distances. This is because viewing from afar usually leads to loss in resolution, which implies fewer detected edges. However, since this lossy behavior affects vertical and horizontal stripes equally, the ratio between E_v and E_h remains almost unchanged.

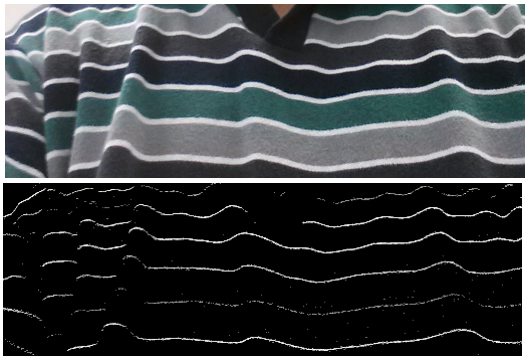


Figure 4: (a) Image for self-fingerprint (b) corresponding energy over wavelet sub-band along horizontal axis.

3.2 Extracting Fingerprints from Glass View

Bob's self-fingerprint is a combination of the spatiogram and wavelet representations. Later, when Alice views Bob through her glass – either from the front or from the back – InSight again crops out a rectangular region around Bob's upper body (below the face/neck), and applies the same fingerprinting operations on this image. These fingerprints – one from Bob and another from Alice – are now ready for matching.

3.3 Fingerprint Matching

Our matching algorithm first computes the spatiogram similarity between each person in Alice's view with the given self-fingerprint (from Bob). Denote the spatiograms to be compared as $S = \{n, \mu, \sigma\}$ and $S' = \{n', \mu', \sigma'\}$, both having B color bins. We define the similarity measure as [13]:

$$\rho = \sum_{b=1}^{B} \sqrt{n_b n'_b}\, 8\pi |\Sigma_b \Sigma'_b|^{1/4}\, \mathcal{N}(\mu_b; \mu'_b, 2(\Sigma_b + \Sigma'_b))$$

[2]Recall that this occurs when the accelerometer senses that the phone is at an appropriate angle, and the user is typing.

Essentially, the similarity decreases (following a Gaussian function) with increasing difference between the colors and their spatial locations.

Following this, we dynamically train a model using the wavelet features of the same two fingerprints. The classifier in use is a bagged decision tree (BDT). The BDT selects random samples from the training data, builds multiple decision trees (each with a subset of samples), and eventually chooses a weighted majority voting result as the final output. The classification results are accompanied by confidence scores that quantify the uncertainty. In the end, the algorithm combines the similarity values from spatiograms with the confidence-scores from wavelet classifiers, and selects a candidate whose confidence exceeds a high threshold. When the confidence is below this threshold, our current system declares "unsure", an attempt to minimize incorrect recognition.

3.4 Refining the Self-Fingerprint

Bob's self-fingerprint is derived from a sliver of his dress, and may not be adequately discriminating against a background of many individuals. Moreover, Alice may view Bob from his back, and this "back fingerprint" may not match well with Bob's self-fingerprint (derived from his front view). This could be due to differing patterns at the back of Bob's shirt; differing wrinkles; and/or unequal lighting conditions. We identify opportunities to consolidate front and back fingerprints, which in turn can improve the robustness of recognizing Bob. Our core intuition exploits natural human mobility as described next.

Consider a social gathering where humans are naturally walking around, to the extent that from any camera view, people in Bob's background changes over time. This *diversity* of backgrounds is likely to become contrasting to Bob at some point. In other words, even if Bob's self-fingerprint is not highly discriminating, in certain favorable situations, the fingerprint may suffice because people in Bob's background are wearing different colored clothes. At this point in time, since Charlie may be able to recognize Bob and actually see his full attire (through her glasses), he can enrich Bob's fingerprint. Enriching would entail informing the cloud that Bob's fingerprint should be updated with spatiogram and wavelet features derived from his trousers, center of the shirt, etc. Later, if Julie happens to view Bob from the back, this enriched fingerprint may now help recognize Bob (perhaps because the trouser colors are discriminating). This can in turn lead to further enrichment – Bob's fingerprint can now be updated with the visual features of his back.

Over time, one may envision everyone's fingerprint getting enriched, which improves recognition, which in turn facilitates enrichment. This recursive process may eventually converge, resulting in fairly unique fingerprints for almost all. Our controlled experiments in the next section will endorse this intuition and indicate room for improvement.

4. EVALUATION

We implement a prototype of InSight using PivotHead camera enabled glasses (Figure 5) and Samsung Galaxy phones running Android. We conduct experiments with 15 users dressed in their regular attires. We explicitly asked these participants to actively use their smartphones. Each phone opportunistically takes "profile" pictures of the user. In this process, InSight selects the most suitable pictures via angle detection using accelerometer readings, face detection, and blur estimation. The automatically chosen pictures are then used to form the "self-fingerprint" for the user. The PivotHead glass was worn by a single user who captured all the other users from the front and from the back. In our preliminary experiments, the users captured in the glass view do not overlap with each other – we controlled the experiment in this manner for the purpose of simplicity.

Our main findings may be summarized as follows: (1) We confirm that color spatiograms and pattern wavelets capture complementary features of a person's dress – together, they are effective in discriminating an individual from the rest 14. (2) When people are facing the glass, they can be accurately recognized using their self-fingerprints. (3) Through *monte carlo* simulations on real fingerprints, we demonstrate how recursive fingerprint refinement can help recognize a person, even when she is facing away from the glass.

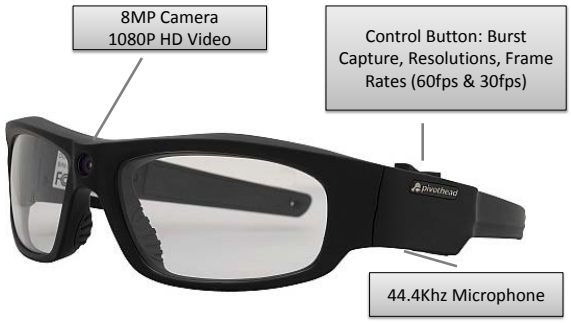


Figure 5: PivotHead camera glasses used for InSight.

4.1 Combining Colors and Patterns

To assess the discriminative abilities of color and pattern features, we first evaluate them separately when 15 people are facing the glass – we extract their features from the glass view. Figure 6(a) shows the confusion matrix represented using a heat map. Element ij of the matrix reflects the similarity score when InSight compares the spatiogram corresponding to person i in the glass-view with that of self-fingerprint of person j. A lighter color indicates higher similarity, and vice versa. If diagonal elements are much lighter than the rest, then spatiograms alone may be considered discriminative. With Figure 6(a), this is true for 80% of the cases.

Figure 6(b) reflects the confidence scores in the confusion matrix, when wavelets are used to extract features from clothing patterns – the recognition accuracy is 73%. While this is not high, we find that spatiograms and wavelets exhibit complementary behavior (compare failure cases in Figure 6(a) and Figure 6(b)). When color spatiograms fail to differentiate two people, pattern wavelets are able to distinguish them. Therefore, we combine these two approaches by computing the product of their similarity and confidence scores. Figure 6(c) presents the result of this hybrid approach. The accuracy improves distinctly; 14 out of 15 people are recognized without ambiguity. The rest of the evaluation employs this hybrid approach.

4.2 Performance with Self-Fingerprints

Since self-fingerprints capture the front view of a person from a close range, its important to characterize whether they are effective when others view the person from a distance, and

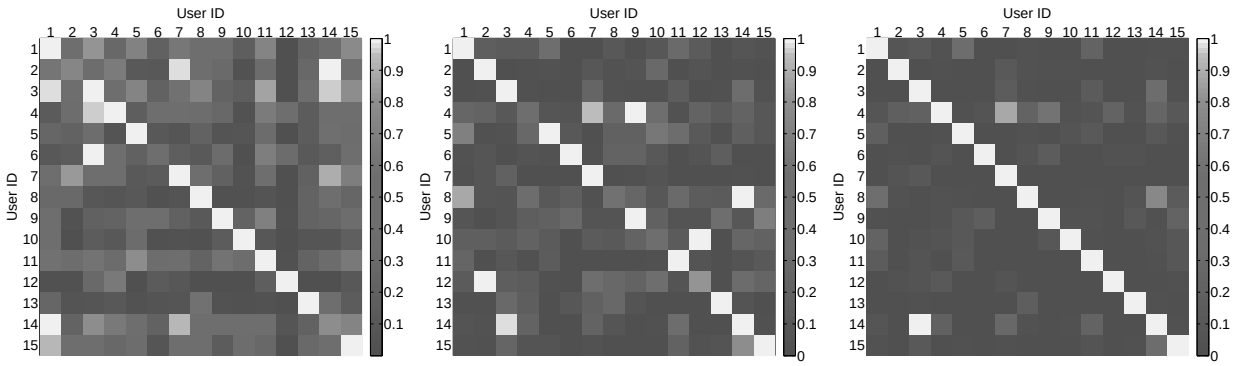


Figure 6: All users facing the glass: (a) Similarity scores from spatiograms; (b) Confidence scores from classification using wavelet features; (c) The effect of combining color with patterns.

sometimes from the back. To address this question, we evaluate the discriminative power of InSight by varying the number of users facing towards and away from the glass. We conduct experiments with *all possible user combinations*, ranging from only 1 user to all 15 users.

Figure 7 evaluates scenarios when all users are facing the glass. For scenarios with n users (on the x axis), the graph shows the average percentage of users correctly recognized, falsely recognized, and unrecognized. The average is computed over all the possible combinations, e.g., 105 combinations in case of 2 users in the view. Evidently, the accuracy drops slightly (from 100% to 93%) from the 1-user to the 15-user scenario. None of the users are recognized incorrectly as someone else. This suggests that when Bob is facing Alice, self-fingerprints may be adequate for reliable human recognition.

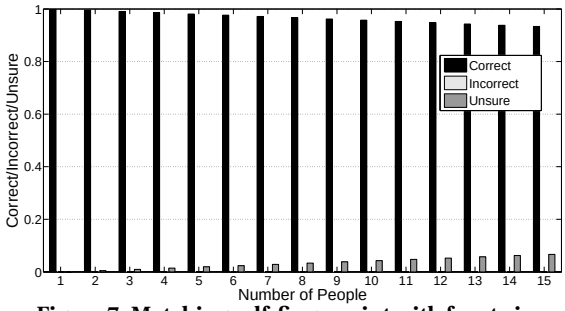


Figure 7: Matching self-fingerprint with front view

Now consider the case where users are facing away from the glass, but their self-fingerprints used to recognize them. Figure 8 shows the results. In most cases, InSight is unable to recognize individuals, particularly when there are many users. This is not too surprising since the glass only captures users' back views and needs to compare them with the self-fingerprints taken from the front. However, a positive outcome is that very few users are incorrectly recognized. Moreover, when there are only few people around, some of them can be recognized from their back view. Next, we describe how these few instances can be leveraged to bootstrap "fingerprint refinement", such that even back-views can discriminate people in a crowded situation.

4.3 Performance with Refined Fingerprints

Consider those few lucky instances when Alice recognizes Bob even though Bob has his back facing Alice (note, these instances are more probable when few people are around). Knowing that

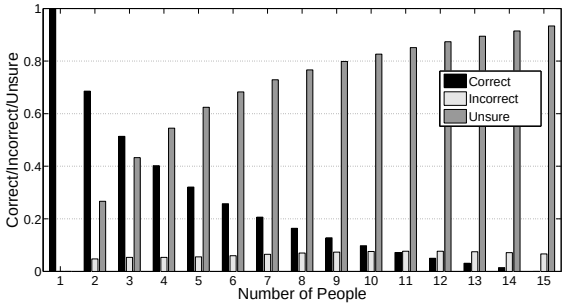


Figure 8: Matching self-fingerprint with back view

this is Bob's back view, InSight can refine Bob's fingerprint (i.e., the cloud updates Bob's self-fingerprint to also contain features from the back-side of his dress). This refinement is feasible only because InSight is rarely wrong in recognizing people (when unsure, InSight refrains from making a recognition). Thus, if Bob is recognized once, Bob can be recognized quite accurately thereafter even in a crowded place. This is regardless of whether his front or back is facing the glass.

To validate the fingerprint refinement approach, we conduct the following monte carlo simulation. We randomly choose 4 people with their backs facing the glass. We compare each of their back views with their self-fingerprints. When there is a strong match, the corresponding <ID, back view> is added to the InSight system. This step is repeated 200 times, and over time more such back views get added to the system. Once the same ID gathers 5 or more back views, we pick the most dominant one as that ID's back fingerprint. Gradually, the accuracy of recognizing a person with a back view improves since it would be compared to back fingerprints when available. Even when Bob's back fingerprint is not in the system, back fingerprints of others help narrow down the search space considerably, enhancing the chances of recognizing Bob using his front-side fingerprint. We perform 500 runs of this simulation and show the average results in Figure 9. The system converges, and increasing number of users get recognized correctly over time, even when all of them have their back facing the glass. Some errors indeed occur, but they do not propagate in the system due to the overwhelming number of correct recognitions.

5. DISCUSSION

Many more challenges need to be addressed, several opportunities need to be exploited. We discuss a few here.

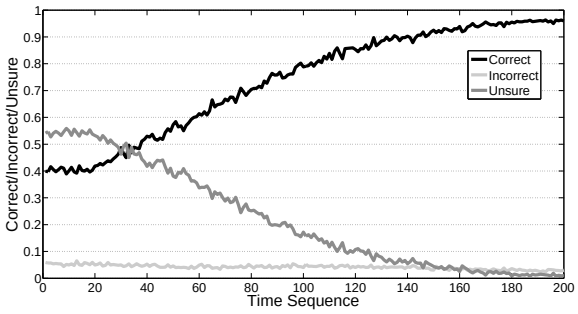


Figure 9: Matching back view after refining fingerprints.

(1) Incremental Deployment. Non-participants of this system – those not running InSight – are likely to be mis-recognized (even though ideally they should be labeled "unknown"). While this is indeed a problem, work arounds may be feasible. If Alice views Charlie but finds that none of the announced fingerprints match with him, then Alice can suspect that Charlie is not a part of the system. Over time, this suspicion could grow as more people are unable to recognize Charlie, eventually tagging Charlie as "unknown". Of course, if Charlie is wearing a dress similar to Bob, then the situation is harder. InSight has to wait for adequate opportunities to find Charlie separate from Bob, so that the suspicion value rises above a threshold – a key challenge for our ongoing work.

(2) Privacy of Opportunistic Pictures. Taking opportunistic pictures, while creating the self-fingerprint, may raise privacy issues. Although the camera takes pictures only at specific orientations, we believe that a concerned user can choose to manually create the self-fingerprint. For instance, if InSight is used occasionally – only at certain conferences or events, or when the user needs to broadcast a message – it may not be burdensome to take a self-picture only at those times. However, regular use may call for additional privacy precautions; one simple way could be to show the automatically-taken picture to the user before using it for fingerprinting.

(3) Overlapping Users in View. When one views a crowded gathering, it may not be easy to crop out each individual from the image. People may be overlapping in their views, and only a small part of their dresses may be visible. InSight will need to evaluate the impact of such complicated views of individuals, especially in crowded settings.

(4) Application scenarios. InSight enables use-cases in which a user Bob intends to convey some information to anyone who visually looks at him. One may view this as a form data broadcast using a visual "source address"; the recipients are all users whose line of sight intersects with the transmitter. Specific instances in which such visual broadcasts are applicable include virtual badges in a conference, students tweeting about their areas of interest in a job fair, etc. Also, several use-cases do not require revealing the user's identity – a person at a basketball stadium can simply tweet "selling an extra ticket for tonight's game". One may even view InSight as a way of social messaging, similar to how people where T-shirts with interesting captions on them.

6. CONCLUSION

This paper pursues a hypothesis that colors and patterns on clothes may pose as a human fingerprint, adequate to discriminate one individual from another in low/moderate density situations. If successful, such a fingerprint could be effectively used towards human recognition or content announcement in the visual vicinity, and more broadly towards enabling human-centric augmented reality. Pivoted on this vision, we develop a proof of concept – *InSight* – using which users create a visual fingerprint of an individual and compare it with that individual's self-created fingerprint. Preliminary evaluation with 15 people wearing natural clothes, suggest promise – we find that clothes indeed exhibit good entropy, and can be automatically fingerprinted/matched with reasonable accuracy. Our ongoing work is focussed on coping with the issue of incremental deployment, as well as exploring motion patterns when visual fingerprints are not unique. We believe there is promise, and are committed to building a fuller, real-time, system.

7. ACKNOWLEDGMENT

We sincerely thank our shepherd, Dr. Stefan Saroiu, as well as the anonymous reviewers for their valuable comments. This work was supported in part by the NSF grant IIS:0910846.

8. REFERENCES

[1] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, and D. Schmalstieg, "Real-time detection and tracking for augmented reality on mobile phones," *IEEE Visualization and Computer Graphics*, 2010.

[2] P. Mistry and P. Maes, "Sixthsense: a wearable gestural interface," in *ACM SIGGRAPH ASIA 2009 Sketches*, 2009, p. 11.

[3] W. Zhao, R. Chellappa, P.J. Phillips, and A. Rosenfeld, "Face recognition: A literature survey," *ACM Computing Surveys (CSUR)*, 2003.

[4] M.A. Turk and A.P. Pentland, "Face recognition using eigenfaces," in *IEEE CVPR*, 1991.

[5] S. Belongie, J. Malik, and J. Puzicha, "Shape matching and object recognition using shape contexts," *IEEE TPAMI*, 2002.

[6] Andrew C. Gallagher and Tsuhan Chen, "Clothing cosegmentation for recognizing people," in *IEEE CVPR*, 2008.

[7] C. Qin, X. Bao, R. Roy Choudhury, and S. Nelakuditi, "Tagsense: a smartphone-based approach to automatic image tagging," in *ACM MobiSys*, 2011.

[8] Ionut et. al. Constandache, "Did you see bob?: human localization using mobile phones," in *ACM*, 2010, MobiCom '10.

[9] A.T. Campbell, S.B. Eisenman, N.D. Lane, E. Miluzzo, R.A. Peterson, H. Lu, X. Zheng, M. Musolesi, K. Fodor, and G.S. Ahn, "The rise of people-centric sensing," *IEEE Internet Computing*, 2008.

[10] S.T. Birchfield and S. Rangarajan, "Spatiograms versus histograms for region-based tracking," in *IEEE CVPR*, 2005.

[11] B.S. Manjunath and W.Y. Ma, "Texture features for browsing and retrieval of image data," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 1996.

[12] Y. Tian and S. Yuan, "Clothes matching for blind and color blind people," *Computers Helping People with Special Needs*, 2010.

[13] CO Conaire, N.E. O Connor, and A.F. Smeaton, "An improved spatiogram similarity measure for robust object localisation," in *IEEE ICASSP*, 2007.

sMFCC : Exploiting Sparseness in Speech for Fast Acoustic Feature Extraction on Mobile Devices – a Feasibility Study

Shahriar Nirjon[1], Robert Dickerson[1], John Stankovic[1], Guobin Shen[2], Xiaofan Jiang[3]
[1]Department of Computer Science, University of Virginia
[2]Microsoft Research Asia, Beijing, China
[3]Intel Labs China, Beijing, China
{smn8z, rfd7a, jas9f}@virginia.edu, jacky.shen@microsoft.com, fred.jiang@intel.com

ABSTRACT

Due to limited processing capability, contemporary smartphones cannot extract frequency domain acoustic features in real-time on the device when the sampling rate is high. We propose a solution to this problem which exploits the sparseness in speech to extract frequency domain acoustic features inside a smartphone in real-time, without requiring any support from a remote server even when the sampling rate is as high as 44.1 KHz. We perform an empirical study to quantify the sparseness in speech recorded on a smartphone and use it to obtain a highly accurate and sparse approximation of a widely used feature of speech called the Mel-Frequency Cepstral Coefficients (MFCC) efficiently. We name the new feature the sparse MFCC or sMFCC, in short. We experimentally determine the trade-offs between the approximation error and the expected speedup of sMFCC. We implement a simple spoken word recognition application using both MFCC and sMFCC features, show that sMFCC is expected to be up to 5.84 times faster and its accuracy is within $1.1\% - 3.9\%$ of that of MFCC, and determine the conditions under which sMFCC runs in real-time.

Keywords

Smartphone, Speech, Sparse FFT, MFCC

1. INTRODUCTION

All major smartphone platforms these days support numerous voice driven applications such as – voice commands (e.g. to launch an application or call some contact), voice-enabled search (e.g. Google's voice search), voice recognizing personal assistant (e.g. iPhone's SiRi), and voice-based biometrics. There are also non-voice sound driven applications, such as the music matching service from Shazam [22]. All of these applications require fast acoustic feature extraction both in time-domain and frequency-domain in order to offer fast, real-time services. While using only the time-domain acoustic features is sufficient in a limited number of applications, the frequency-domain features are a must for a robust and accurate encoding of acoustic signals.

State-of-the-art smartphone applications and platforms that extract acoustic features are primarily of two kinds. The first kind records the audio and sends it to a remote server over the Internet for further processing. This method has several limitations such as the requirement for an uninterrupted Internet connectivity and high bandwidth, and the associated expense of sending a large chunk of audio data over the cellular network. The second kind, on the other hand, performs the entire signal processing task inside the phone. But the limitation of this approach is that in order for fast and real-time feature extractions, they must limit the sampling rate to the minimum. For example – SpeakerSense [14] and SoundSense [15] limit their maximum sampling rate to 8 KHz. Hence, the quality of sampled speech suffers from the aliasing problem and the extracted features are often of low quality [2]. Although a sampling rate of 8 KHz satisfies the Nyquist criteria for human speech ($300 - 3300$ Hz), practically the higher the sampling rate the better it is in producing high quality samples. Furthermore, for non-speech acoustic analysis, a sampling rate of 44.1 KHz is required to capture the range of frequencies in human hearing ($20 - 20000$ Hz). But the problem is – there is no efficient algorithm that extracts frequency domain acoustic features inside the phone in real-time at such high sampling rates.

In this paper, we propose a novel solution to this problem which enables the extraction of frequency domain acoustic features inside a smartphone in real-time, without requiring any support from a remote server even when the sampling rate is as high as 44.1 KHz. We are inspired by a recent work [9, 8] coined sparse Fast Fourier Transform (sFFT) – which is a probabilistic algorithm for obtaining the Fourier Transformation of time-domain signals that are sparse in the frequency domain. The algorithm is faster than the fastest Fourier Transformation algorithm under certain conditions. Our goal in this paper is to analyze the feasibility of applying the sFFT to extract a highly accurate and sparse approximation of a widely used feature for speech, called the Mel-Frequency Cepstral Coefficients (MFCC) on the phone. Besides speech recognition, MFCC features are widely used in many other problems such as speaker identification [19], audio similarity measure [10], music information retrieval [13],

ACM HotMobile'13, February 26–27, 2013, Jekyll Island, Georgia, USA.

and music genre classification. However, we limit our scope in this work to speech data only.

We perform an empirical study involving 10 participants (5 male and 5 female participants) where we collect more than 350 utterances of speech per person, recorded at different sampling rates. In our study, we quantify the sparseness of speech and show that human voice is suitable for applying sFFT to compute frequency domain acoustic features efficiently. We analyze the sensitivity of sFFT in approximating MFCC, and based on this, we design an algorithm to efficiently extract MFCC features using the sFFT instead of the traditional FFT. We name this new feature the sparse MFCC or sMFCC, in short. We experimentally determine the trade-offs between the approximation error and the expected speedup of sMFCC.

As a proof of concept, we implement a simple spoken word recognition application using both MFCC and sMFCC features and compare their accuracy and expected running time on our collected data. As the research is still in progress, we only implement the MFCC feature extraction on the smartphone while the rest of the analysis done on a PC. In future, we plan to complete the porting of the entire app to the smartphone. We also plan to explore other types of applications that require general purpose acoustic feature extraction on smartphones. The main contributions of this paper are:

- A study on 10 smartphone users to quantify the sparseness of speech data recorded on smartphones and to analyze the feasibility of using sFFT for frequency domain feature extraction.
- We describe an efficient algorithm for computing a highly accurate and sparse approximation of MFCC features which exploits the sparseness in speech.
- We implement a simple spoken word recognition application using both MFCC and sMFCC features, show that sMFCC is expected to be upto 5.84 times faster and its accuracy is within $1.1\% - 3.9\%$ of that of MFCC, and determine the conditions under which sMFCC runs in real-time.

2. BACKGROUND

2.1 Mel-Frequency Cepstral Coefficients

The Mel-Frequency Cestrum Coefficients (MFCC) is one of the most popular short-term, frequency domain acoustic features of speech signals [4]. The MFCC have been widely used in speech analysis because of their compact representation (typically, each speech frame is represented by a 39-element vector), close resemblance to how human ear responds to different sound frequencies, and their less susceptibility to environmental noise.

The MFCC feature extraction starts with the estimation of power spectrum which is obtained by taking the square of the absolute values of the FFT coefficients. However, prior to computing the power spectrum, typically each speech frame passes through a pre-emphasis filter which is followed by a windowing process. The log-power spectrum is used instead of the power-spectrum as human hearing works in decibel scales. The log-power spectrum then goes through a filtering process. A filter-bank with around 20 triangular band-pass filters is applied to reduce the dimensionality. These filters follow a Mel-scale which is linear up to 1 KHz and logarithmic for the larger frequencies – resembling human hearing. Finally, a discrete cosine transform (DCT) is performed to compress the information and to make the vectors uncorrelated. Only the lower-order coefficients (typically 13) are used and the rest are discarded. The 13-MFCCs plus the deltas and double deltas constitute a 39-element feature vector for each speech frame.

2.2 Sparse Fast Fourier Transform

The discrete Fourier transform (DFT) is one of the most significant algorithms in the digital signal processing domain. The fastest algorithm that computes DFT of an n-dimensional signal is $O(n \log n)$-time. However, a recent algorithm [9, 8] coined sparse FFT (sFFT) has broken this bound for a special case of DFT where the signals are sparse. A signal is considered sparse if most of its Fourier coefficients are small or very close to zero. For a small number k of non-zero Fourier coefficients, sFFT computes the Fourier transformation in $O(k \log n)$-time.

The basic idea of sFFT is to hash the Fourier coefficients into a small number of bins. The signal being sparse in the frequency domain, it is less likely that each bin will have more than one large coefficient. The binning process is done in $O(B \log n)$ where B is the number of bins – by at first permuting the time-domain signals and then filtering them. Each bin at this point ideally has only one large Fourier coefficient, and only such 'lonely' coefficients are taken into the solution. The process is repeated $\Theta(\log k)$ times, each time varying the bin size $B = k/2^r$ where the integer $r \in [0, \log k]$, so that all k coefficients are obtained. The overall running time of the algorithm is dominated by the first iteration, and hence the time complexity is $O(k \log n)$. The algorithm is probabilistic, but for exact k-sparse signals (i.e. at most k of the coefficients are significant), the algorithm is optimal, as long as $k = n^{\Omega(1)}$.

3. MOTIVATION

Smartphones allow a fixed number of sampling rates to capture raw audio signals from the microphone. Ideally the choice of an appropriate sampling rate should be driven by the application's QoS requirements. But often the developers are forced to choose a lower sampling rate than the desired one due to the limited processing power of the device. For example, in general-purpose acoustic processing, a 44.1 KHz Nyquist sampling rate is required to capture the range of frequencies in human hearing ($20 - 20000$ Hz). Even in speech processing problems, oversampling at 16 KHz helps avoid aliasing, improves resolution and reduces noise [23]. But at higher sampling rates, the real-time performance of the smartphone gets worse and the developers are forced to select the minimum rate compromising the quality of sampled speech.

Figure 1 compares the time to compute MFCC feature vectors from audio records of different durations at 3 different sampling rates. The experiment is done on a Nexus S smartphone running Android 2.3 that supports 8 KHz, 22.05 KHz, and 44.1 KHz sampling rates. We see that, the time to compute MFCC features is always longer than the duration of the recorded audio when the sampling rate is higher than 8 KHz. For example, the computation of MFCC vectors of a 4-second recording takes on average 7.02 s at 22.05 KHz, and 11.85 s at 44.1 KHz. Therefore, at these higher rates, the application is not capable of real-time performance.

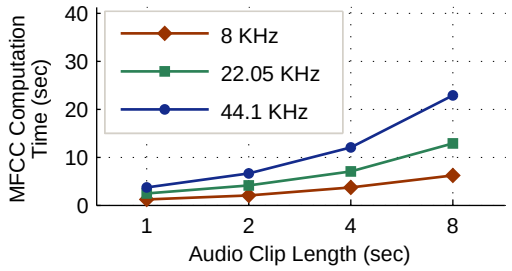


Figure 1: The MFCC computation time is $2-4$ times longer than the audio clip length when the sampling rate is high.

Our goal is therefore to investigate the problem: whether or not it is possible to compute MFCC feature vectors in real-time on a smartphone when the data rate is high? In an attempt to answer this question, we study the nature of human speech recorded on a smartphone. We hypothesize that the sparseness in speech can be exploited to compute a close approximation of MFCC feature vectors on a smartphone in real-time. While the focus of this work is on speech, an analysis of the general-purpose acoustic signals based on similar principles is under investigation and we leave it as our future work.

4. THE SPARSE MFCC ALGORITHM

The idea of sparse MFCC algorithm is to compute a sparse approximation of MFCC features from a given frame of discrete time-domain signals x_n of length n. The algorithm uses a modified version of sFFT as a subroutine. We denote the new feature by sMFCC to signify its relation to sFFT. Like the sFFT to FFT, sMFCC is an approximation to MFCC, where the approximation error is defined by,

$$error(k) = 1 - \mathbf{M\hat{F}CC} \cdot \mathbf{sM\hat{F}C\hat{C}C(k)} \quad (1)$$

where, $\mathbf{M\hat{F}CC}$ and $\mathbf{sM\hat{F}C\hat{C}C(k)}$ are unit vectors, and their scalar product is subtracted from unity to obtain the approximation error. sMFCC is expressed as a function of the sparseness parameter k, which is one of the key parameters to the sFFT algorithm. The following two sections describe the sMFCC extraction algorithm in detail.

4.1 Estimation of Sparseness

Since the value of k is a key input to the sFFT algorithm and sFFT is used in our computation, the first step of sMFCC algorithm is to find the optimum value of k, denoted by k^*, for which the MFCC approximation error is within a small, non-negative threshold δ, i.e.,

$$k^* = \min_{\text{error(k)}<\delta} k \quad (2)$$

In order to obtain k^*, we first compute the MFCC using the standard FFT algorithm which runs in $O(n \log n)$. We then perform an iterative $O(n)$ search for $k \in [1, n]$ until we find the optimum k^*. The computation of sMFCC(k), for $k \in [1,\ n]$, is optimized by precomputation. We precompute the FFT, keep the FFT coefficients sorted in non-increasing order, and take only the largest k coefficients while making other coefficients zero – while computing sMFCC(k). Note that, this step of our algorithm does not use sFFT and runs in $O(n \log n)$. The shape of the function $error(k)$ (Figure 4 in Section 6.2) however suggests that, instead of a linear search over all values of k, we could expedite the process with a binary search. However, k is estimated once per utterance, i.e. using the first $5-10$ frames once voice activity is detected, and hence the amortized cost of this step is not significant.

4.2 Computing sMFCC

Once we obtain the sparsity parameter k, we compute the sMFCC for each frame in 3 steps which we describe next.

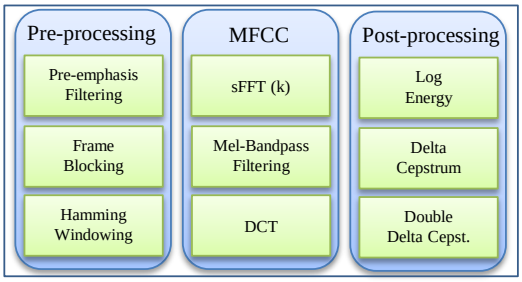


Figure 2: The computational tasks involved in the sMFCC feature extraction process is shown.

Pre-processing: The time-domain signals are first passed through a high-pass pre-emphasis filter (Eq. 3) to amplify the high-frequency formants that are suppressed in speech. We then segment the signals into frames of 64 ms with an overlap of 1/3 of the frame size. A hamming window (Eq. 4) is applied to each frame to ensure the continuity between the first and last points which is required for FFT.

$$x[i] = x[i] - 0.95 \times x[i-1] \quad (3)$$

$$hamm(i) = 0.54 - 0.46\cos(\frac{2\pi i}{n-1}) \quad (4)$$

MFCC: We modify the sFFT algorithm to fit into our algorithm. Recall that, sFFT tries to extract k Fourier coefficients in $\log k$ iterations to guarantee the retrieval of all k coefficients. However, in our experience, sFFT returns most (at least 75%) of the k coefficients in a single iteration. We, therefore, modify the sFFT by running it for only a single iteration with a slightly larger sparseness parameter of $k = \min(n, \lceil 4k^*/3 \rceil)$ to speed up the process. Once the Fourier coefficients are obtained, we follow the standard procedure of MFCC [5]. We apply 20 triangular band-pass filters (called Mel-banking) to obtain 20 log energy terms, perform a DCT to compress them, and take the first 13 coefficients to constitute a 13-element sMFCC vector M.

Post-processing: The 13-element sMFCC vector is augmented to include the delta and double delta cepstrums to add dynamic information into the feature vector, and thus we obtain a 39-element feature vector.The deltas Δ and double deltas Δ^2 are computed using the following two equations,

$$\Delta_i = M_{i+2} - M_{i-2} \quad (5)$$

$$\Delta_i^2 = M_{i+3} - M_{i-1} - M_{i+1} + M_{i-3} \quad (6)$$

5. EXPERIMENTAL SETUP

We perform an empirical study involving 10 volunteers, in which, we record their speech using a smartphone in home environments. Each participant was given a list of 86 English words and a paragraph from a book. The wordlist includes 10 digits, 26 characters of the English alphabet, 25 mono-syllable and 25 poly-syllable words. Participants were asked to utter each word 4-times – clearly and at regular pace. There were about a 2-second gap between two spoken words so that we could extract and model each word separately. The group of participants is comprised of undergraduate and graduate students, researchers, professionals, and their family members. Their ages are in the range of $20-60$, and they have diversities in speaking style and ethnicity. The smartphone we used during the data collection is a Nexus S phone running Android 2.3.6 OS. It has a 1 GHz Cortex A8 processor, 512 MB RAM, 1 GB internal storage, and 13.31 GB USB storage. The execution time of MFCC is measured on the smartphone using Android's API, and speedup of sMFCC is the ratio of running times of MFCC and sMFCC.

6. EXPERIMENTAL RESULTS

We conduct four sets of experiments. First, we quantify the sparseness of speech in our empirical dataset. Second, we show the approximation error in sMFCC. Third, we establish the condition for speedup in sMFCC. Finally, we describe a simple spoken word recognizer to quantify the cost and benefits of sMFCC over MFCC.

6.1 Sparseness in Speech

The sparseness of signal is defined by the number of negligible Fourier coefficients in its spectrum. A coefficient is considered negligible if it contains a very small amount of signal power. Sparseness in audio signals depends on the audio type. In this paper, we study clean speech signals only.

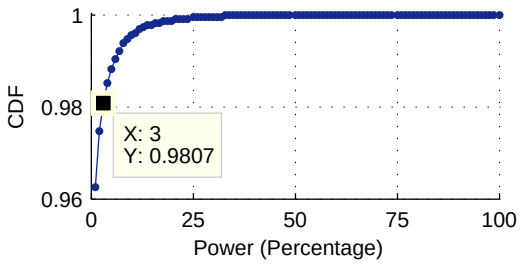


Figure 3: 98.07% of the Fourier coefficients in our dataset contains only 3% or less power.

Figure 3 shows the cumulative distribution function (CDF) of power in the Fourier spectrum of the utterances in our dataset. To obtain this plot, we compute the FFT of all the utterances in our dataset, take the squared magnitude of FFT to obtain the signal power, construct a 100-bin histogram where each bin corresponds to a range of powers, compute the fraction of Fourier coefficients that are in each bin, and compute the CDF. Each point on the plot tells us, what fraction of the signals have power less than or equal to the range corresponding to the X-coordinate. For example, the marked point on the plot denotes that 98.07% of the Fourier coefficients in each utterance of our dataset contains only 3% or less power. The rest 1.93% coefficients that are significant are permuted (see [9, 8] for the details) in the frequency domain so that the spectrum becomes extremely sparse and ideal for the application of sFFT.

6.2 Sparse Approximation Error in sMFCC

The quality of sMFCC features depends on the choice of an appropriate k. The larger the value of k, the closer it is in approximating MFCC. In this experiment, we analyze the sensitivity of k to the MFCC approximation error.

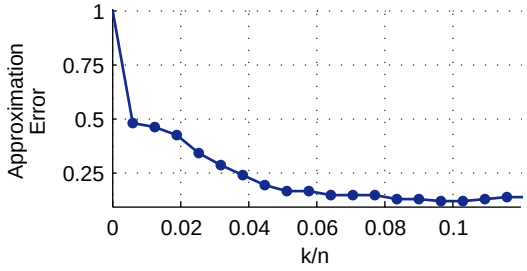


Figure 4: The higher the value of k, the better approximation of MFCC we get.

Figure 4 shows the approximation error for the range of sparseness $k \in [0.00625, 0.125]$. We consider this range since it contains most of the significant Fourier coefficients and is also important for the discussion of speedup in Section 6.3. Each point on the plot corresponds to the mean approximation error of sMFCC for a given k, where the mean is taken over all 64 ms frames in all the utterances in our dataset. The frame size is $n = 4096$ samples, which is the next power of 2 that holds a 64 ms frame at 44.1 KHz. This figure guides us in choosing the parameter k in our sMFCC computation algorithm if we want to keep the MFCC approximation error below a desired threshold. A very close approximation ($< 1\%$ error) is possible by choosing $k/n = 0.2$ or higher. However, such close approximation may not be required in an actual application which we will see in Section 6.4.1. The reason is that human speech being sparse, even at a smaller k/n ratio, the absolute value of approximation error is not high.

6.3 Speedup in sMFCC

Sparseness in speech is the source of expected speedup in sFFT and hence in sMFCC as well. Figure 5 shows the speedup in sMFCC for the range of sparseness $k \in [0.00625, 0.125]$. We observe the maximum speedup of 5.84 when the sparsity parameter is at its minimum. As we consider more and more FFT coefficients while computing sMFCC, the speedup decreases and after $k/n = 6.769\%$ the regular MFCC becomes faster than its sparse counterpart. This limitation comes from the fundamental bound of sFFT, which says, sFFT is faster than FFT when $k/n < 3\%$ [8]. However, our modified version of sFFT is faster for the reason we discussed earlier in Section 4.2, and hence we have a larger bound of 6.796%.

6.4 A Simple Spoken Word Recognizer

The goal of this experiment is to analyze the tradeoff between the accuracy and expected speedup of sMFCC features in an application scenario. To do so, we implement a

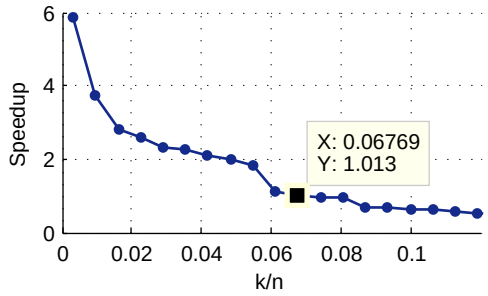


Figure 5: sMFCC has a better running time than MFCC as long as the sparseness $k/n < 6.769\%$.

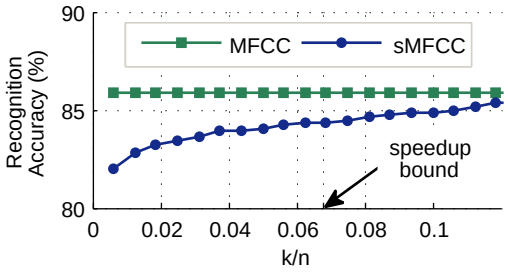


Figure 6: The recognition accuracy of sMFCC-based classifier is within $1.1\% - 3.9\%$ of the MFCC-based one inside the speedup zone.

simple spoken word recognizer that is essentially a speech-to-text program for a single word from a fixed vocabulary. The recognizer consists of two parts: a smartphone app and a word recognizer running on a PC. The word recognizer running on a PC is for proof of concept, our envisioned use-case is however to run it on the phone.

At first, the user turns on the application on the phone and presses the 'speak now' button. The smartphone then starts sampling the microphone at 44.1 KHz and keeps producing speech frames until the user presses the 'stop' button. Each frame goes through the MFCC feature extraction process which happens in real-time. Each spoken word produces a number of frames, and a 39-element MFCC feature vector is obtained for each frame. We take the mean and the standard deviation of each of the 39 MFCC coefficients over all frames to obtain a single 78-element feature vector which is used in the classification step. The feature vectors are then sent to a PC for classification and further analysis. The ground truth is obtained by taking notes manually. We train a Support Vector Machine (SVM) classifier in order to recognize the words. A 3-fold cross validation is used to determine the accuracy of the classifier where 75% of each user's data is taken for training and the rest is used for validation.

6.4.1 Accuracy

We compare the accuracy of the sMFCC-based SVM classifier with that of the MFCC-based one. The baseline MFCC-based classifier is essentially a special case of sMFCC-based one with a sparseness $k/n = 1$, and has a recognition accuracy of 85.85%. Figure 6 compares the recognition accuracy of sMFCC-based classifier to the baseline for the same range of k/n we have been using throughout the paper. We observe that, the accuracy of sMFCC-based classifier is initially 3.9% lower than the baseline, and the two accuracies becomes practically identical once k/n reaches 0.12. However, from the discussion in Section 6.3 we know that, sMFCC runs faster than MFCC only when the k/n ratio is within the 6.679% bound. For this boundary case, sMFCC shows an accuracy of 84.75%, which is only 1.1% lower than the baseline. In summary, with sMFCC-based classifier for our simple word recognition problem, we can achieve a faster running time than the baseline with a very small (1.1%-3.9%) sacrifice in recognition accuracy.

6.4.2 Computation Time

The MFCC feature extraction process runs once per spoken word. Hence, the execution time depends on the duration of the spoken word which varies from person-to-person and from word-to-word. In our dataset, the duration of speech ranges from the minimum of 400 ms to the maximum of 2.88 s. We, therefore, compute the expected feature extraction time $E[\phi_{feat}(d_i)]$ using the following equation,

$$E[\phi_{feat}(d_i)] = \sum f_i \times \phi_{feat}(d_i) \quad (7)$$

where, d_i is the duration of speech, f_i is the frequency of utterances with duration d_i, and $\phi_{feat}(d_i)$ is the feature extraction time (either MFCC or sMFCC).

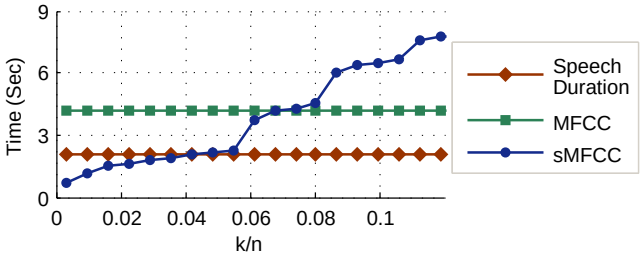


Figure 7: The sMFCC feature extraction algorithm runs in real-time as long as $k/n < 4.835\%$.

Figure 7 shows the mean speech duration and the expected computation times of MFCC and sMFCC feature extraction process. We see that, the expected MFCC computation time is 4.21 s, which is about 2 times higher than the duration of speech (2.11 s). On the other hand, the expected computation time for sMFCC varies with k/n: it increases as k/n increases, is lower than the duration of speech as long as $k/n < 4.835\%$, and crosses the MFCC computation time when k/n reaches the speedup limit of 6.679%. Hence, applications that require fast and real-time feature extraction should set the k parameter such that k/n is below the real-time limit of 4.835%. At this limit, the accuracy of the sMFCC-based word recognizer is 83.97%, which is only 1.88% lower than the accuracy of the baseline MFCC-based classifier.

Discussion: We show the tradeoff between accuracy and speedup for single word recognition problem from a limited vocabulary. The developer of the app should decide what k/n value to pick for his application. For different apps, the most suitable value of k/n will be different, and need to be chosen from a similar tradeoff curve.

7. RELATED WORK

Sparseness in data is exploited in many application domains such as learning decision trees [12], compressed sens-

ing [6], analysis of Boolean functions [21], large scale time series data analysis [18], similarity search [1], and homogeneous multi-scale problems [3]. In our work, we perform an empirical study on the sparseness in speech data collected on smartphones with the goal of exploiting the sparseness to expedite the MFCC feature computation.

MFCC is a widely used feature for analyzing acoustic signals [5, 17, 19, 13, 10]. MFCC is used for spoken word recognition [5], voice recognition [17], speaker identification [19], music modeling [13], and music similarity measure [10]. [7] presents a nice comparison of different MFCC implementations. However, in our work, we introduce a sparse MFCC (sMFCC) which is a sparse approximation of MFCC and is efficient to extract.

Comparison of several speech recognition techniques on mobile devices are described in [11]. [14] performs speaker identification, [15] classifies sound into voice, music or ambient sound, and [16] classifies conversion. But all of these applications limit their sampling rate to its minimum. [20] uses a high sampling rate to extract heart beats from acoustic signal, but the system is not fully real-time. In this work, we analyze the feasibility of using sMFCC features that is expected to run in real-time and at higher data rates.

8. CONCLUSION AND FUTURE WORK

In this work, we propose an algorithm that exploits sparseness in speech to extract a highly accurate and sparse approximation of MFCC features (sMFCC) efficiently inside a smartphone in real-time when the sampling rate is as high as 44.1 KHz. We implement a simple spoken word recognition application using both MFCC and sMFCC features, show that sMFCC is upto 5.84 times faster than MFCC and its accuracy is within $1.1\% - 3.9\%$ of that of MFCC, and determine the conditions under which sMFCC runs in real-time.

Our future work includes three main directions. First, we plan to improve our empirical study by adding more participants, investigating the continuous speech recognition problem rather than just single word recognition, incorporating more complex classifiers, and considering background noise. Second, we plan to complete porting the entire application to the smartphone which includes on-line training and classification modules. Third, we plan to explore other types of sounds such as music and environmental sounds, and other acoustic features.

Acknowledgments

This work was supported, in part, by NSF Grant EECS-0901686.

9. REFERENCES

[1] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. In *International Conference on Foundations of Data Organization and Algorithms*, pages 69–84, 1993.

[2] K. Brandenburg. *Perceptual Coding of High Quality Digital Audio*, volume 437. Springer US, 2002.

[3] I. Daubechies, O. Runborg, and J. Zou. A sparse spectral method for homogenization multiscale problems. *Multi-scale Modeling Simulation*, 6(3):711–740, 2007.

[4] S. Davis and P. Mermelstein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 28(4):357 – 366, aug 1980.

[5] S. Davis and P. Mermelstein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 28(4):357 – 366, aug 1980.

[6] D. Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52(4):1289–1306, 2006.

[7] Z. Fang, Z. Guoliang, and S. Zhanjiang. Comparison of different implementations of mfcc. *Journal of Computer Science and Technology*, 16(6):582–589, nov 2001.

[8] H. Hassanieh, P. Indyk, D. Katabi, and E. Price. Nearly optimal sparse fourier transform. In *44th ACM Symposium on Theory of Computing 2012 (STOC '12)*, NewYork, NY.

[9] H. Hassanieh, P. Indyk, D. Katabi, and E. Price. Simple and practical algorithm for sparse fourier transform. In *ACM-SIAM Symposium on Discrete Algorithms 2012 (SODA '12)*, Kyoto, Japan.

[10] J. Jensen, M. Christensen, M. Murthi, and S. Jensen. Evaluation of mfcc estimation techniques for music similarity. In *European Signal Processing Conference 2006 (EUSIPCO '06)*.

[11] A. Kumar, A. Tewari, S. Horrigan, M. Kam, F. Metze, and J. Canny. Rethinking speech recognition on mobile devices. In *2nd International Workshop on International User Interfaces for Developing Regions (IUI4DR)*, Palo Alto, CA, 2011.

[12] E. Kushilevitz and Y. Mansour. Learning decision trees using the fourier spectrum. In *ACM Symposium on Theory of Computing 1991 (STOC '91)*.

[13] B. Logan. Mel frequency cepstral coefficients for music modeling. In *International Symposium on Music Information Retrieval 2000 (ISMIR '10)*.

[14] H. Lu, A. J. B. Brush, B. Priyantha, A. K. Karlson, and J. Liu. Speakersense: energy efficient unobtrusive speaker identification on mobile phones. In *9th International Conference on Pervasive Computing 2011 (Pervasive '11)*, pages 188–205, San Francisco, CA.

[15] H. Lu, W. Pan, N. D. Lane, T. Choudhury, and A. T. Campbell. Soundsense: scalable sound sensing for people-centric applications on mobile phones. In *7th International Conference on Mobile Systems, Applications, and Services 2009 (MobiSys '09)*, pages 165–178, Poland.

[16] E. Miluzzo, N. D. Lane, K. Fodor, R. Peterson, H. Lu, M. Musolesi, S. B. Eisenman, X. Zheng, and A. T. Campbell. Sensing meets mobile social networks: the design, implementation and evaluation of the cenceme application. In *6th ACM conference on Embedded network sensor systems 2008 (SenSys '08)*, pages 337–350, Raleigh, NC, USA.

[17] L. Muda, M. Begam, and I. Elamvazuthi. Voice recognition algorithms using mel frequency cepstral coefficient (mfcc) and dynamic time warping (dtw) techniques. *Journal of Computing*, 2(3):138–143, 2010.

[18] A. Mueen, S. Nath, and J. Liu. Fast approximate correlation for massive time series data. In *ACM SIGMOD International Conference on Management of Data 2012 (SIGMOD '10)*, pages 171–182, Indiana, USA.

[19] K. Murty and B. Yegnanarayana. Combining evidence from residual phase and mfcc features for speaker recognition. *Signal Processing Letters, IEEE*, 13(1):52 – 55, jan. 2006.

[20] S. Nirjon, R. Dickerson, Q. Li, P. Asare, J. Stankovic, D. Hong, B. Zhang, G. Shen, X. Jiang, and F. Zhao. Musicalheart: A hearty way of listening to music. In *10th ACM Conference on Embedded Networked Sensor Systems 2012 (SenSys '12)*, Toronto, Canada.

[21] R. O'Donnell. Some topics in analysis of boolean functions (tutorial). In *ACM Symposium on Theory of Computing 2008 (STOC '08)*.

[22] A. Wang. An industrial-strength audio search algorithm. In *International Symposium on Music Information Retrieval 2003 (ISMIR '03)*.

[23] J. Watkinson. *The Art of Digital Audio*. Newton, MA, USA, 1993.

Lowering the Barriers to Large-Scale Mobile Crowdsensing

Yu Xiao
Carnegie Mellon University
Aalto University
yuxiao@cs.cmu.edu

Pieter Simoens
Carnegie Mellon University
Ghent University - iMinds
pieter.simoens@intec.ugent.be

Padmanabhan Pillai
Intel Labs
padmanabhan.s.pillai@intel.com

Kiryong Ha
Carnegie Mellon University
krha@cmu.edu

Mahadev Satyanarayanan
Carnegie Mellon University
satya@cs.cmu.edu

ABSTRACT

Mobile crowdsensing is becoming a vital technique for environment monitoring, infrastructure management, and social computing. However, deploying mobile crowdsensing applications in large-scale environments is not a trivial task. It creates a tremendous burden on application developers as well as mobile users. In this paper we try to reveal the barriers hampering the scale-up of mobile crowdsensing applications, and to offer our initial thoughts on the potential solutions to lowering the barriers.

1. INTRODUCTION

In recent years, there has been phenomenal growth in the richness and diversity of sensors on smartphones. It is now common to find two cameras, a GPS module, an accelerometer, a digital compass, a gyroscope and a light sensor in a single smartphone. And there is more to come! The rich information about the smartphone user's activity and environment provided by these sensors inspired the first wave of sensing applications that personalized user experience based on the sensed context. Now, a second wave of mobile sensing applications is gaining momentum. The focus has shifted from individual sensing towards *crowdsensing,* defined as "individuals with sensing and computing devices collectively sharing information to measure and map phenoma of common interest" [10]. Initially, crowdsensed inputs were analyzed offline, for example in the analysis of transportation activities in urban spaces [34], for the measurement of interperson similarity [14], or for mental and physical health assessment of elder people [23]. In more recent crowdsensing applications, the collected inputs are processed in real time. Examples include traffic monitoring [35, 36], public safety management [27], and collaborative searching [31].

A hypothetical use case serves to illustrate the potential benefits of crowdsensing using information-rich multimedia sensors and some potential pitfalls [25]. Imagine that a small child gets lost while watching a parade in the middle of a large city. The distraught parents, upon noticing their child is missing, immediately use their smartphone to initiate a search, providing sample images with their child's face. A crowdsensing search application tries to match these with the videos and images being captured by the many smartphone cameras in the crowd. Any potential matches are forwarded to the parents' phone, along with GPS location information. With thousands of electronic eyes applied to this problem, the child is quickly found, before she herself is even aware of being lost. For this use case, the large number of smartphone cameras in use makes it likely the child appears in one or more captured images; however, the crowdsensing search application itself can succeed only if a sufficiently large number of smartphone users participate.

More generally, there is a growing realization that *scale* is the key to the success of crowdsensing applications. Since individual users may go offline and individual sensor readings may be inaccurate or corrupted, the reliability and trustworthiness of crowdsensing applications scales more than proportionally with the number of users. Access to a vast user base is thus crucial. However, our survey of the literature shows that today's mobile crowdsensing applications using physical sensors like GPS have rarely been scaled up to more than 1,000 participants.

Table 1 shows a representative sample of crowdsensing studies. Much to our surprise, the number of participants is often omitted in the papers reporting these studies. When concrete numbers are provided, the crowd sizes are usually small. It is only with data sources that are easy to collect (e.g. from social networking applications such as Twitter) that larger crowds have been studied. The one notable exception is the work of Balan et al [2], discussed in Section 2.

What limits the scaling of crowdsensing applications? In this paper, we explore this issue and and propose an architectural solution. We then explore the merits of this architecture, and discuss potential implementation challenges.

2. OBSTACLES TO CROWD SCALING

Crowdsensing applications, including the ones that exist today and the emerging class of applications making use of richer multimedia sensors, face three major barriers to achieving the large crowd sizes critical to their success.

The first obstacle is the heterogeneity of sensing hardware and mobile platforms. In today's mobile device market, there are at least three popular software platforms, includ-

HotMobile'13, February 26–27, 2013, Jekyll Island, Georgia, USA.

Reference	Mobile Platform	Application Category	Crowd Size	Input
Zhou et al. [35] (Mobisys 2012)	Android	Transportation	unknown	cell tower ID, audio signal accelerometer
Tiramisu [36] (CHI 2011)	iOS	Transportation	28	GPS
SignalGuru [12] (MobiSys 2011)	iOS	Transportation	13	video frames
Balan et al. [2](Mobisys 2011)	Car GPS	Transportation	15000	GPS
Mathur et al. [16] (Mobisys 2010)	Car GPS	Transportation	500	GPS
Niu et al. [20] (Com.geo 2011)	Blackberry	Transportation	unknown	GPS
Bao et al. [3] (Mobisys 2010)	Symbian, iPod	Social Application	unknown	video
Wirz et al. [30] (SCI 2011)	Android	Social Application	unknown	GPS
CrowdSearch [31] (Mobisys 2010)	iOS	Search	unknown	image
GeoLife [34] (WWW 2009)	GPS phones	User Behavior Study	107	GPS
SoundSense [15] (Mobisys 2009)	iOS	User Behavior Study	unknown	audio stream
#EpicPlay [28] (CHI 2012)	Twitter	Social Application	unknown	tweets
Wakamiya et al. [29] (ICUIMC 2012)	Twitter	User Behavior Study	39898	tweets
Fujisaka et al. [9](ICUIMC 2012)	Twitter	User Behavior Study	8139	tweets
CrowdSearcher [5] (WWW 2012)	Facebook	Search	137	text

Table 1: Representative Sample of Crowd-sensing Applications

ing Android, iOS and Windows 8. Applications written for any of these can not be run on the others. Even different versions of a particular platform are sometimes incompatible, due to changes in hardware or evolution of the software APIs. Furthermore, the Apps model in vogue today, along with the relatively low processing power of mobile devices, has encouraged smaller, stand-alone applications, and discouraged the development of external libraries, middleware, and virtualization techniques to bridge the differences between platforms. There is no sign that a single platform will dominate this fragmented market in near future. For true ubiquity, application developers need to write, test, support, and maintain versions of their applications for all of these platforms. In a sense, the complexity of the crowdsensing application space grows with cross product of the number of platforms and the number of applications.

This issue of heterogeneity is underscored by the experience of Balan et al. [2], who conducted one of the largest crowdsensing studies to date. It took them six months to deploy one version of their GPS-based crowdsensing application on 15,000 taxis in Singapore, mainly due to the heterogeneity of the on-car GPS devices provided by different vendors. Web-based applications implemented in HTML5 are sometimes put forward as a "write once, run everywhere" alternative. Unfortunately, the HTML5 sensor APIs available on mobile browsers are still quite limited and the support for different sensors varies from browser to browser and from platform to platform. While geolocation tracking using GPS is widely supported in the up-to-date mobile browsers, accessing the microphone and video cameras is not possible in most cases [17]. In practice, due to the potential incompatibility between browsers (e.g. the inconsistent support for audio and video codecs), developers still have to customize their code for different browsers in some cases.

The second obstacle is the burden today's crowdsensing applications place on users. Today, each user must install a separate proprietary application for every crowdsensed experiment in which s/he wishes to participate. As a result, the deployment of a single crowdsensing application is limited by the rate at which users adopt and install it on their devices. It can take weeks or months for a newly introduced application to reach the critcial mass of participants needed for it to be useful. Rapid, large-scale deployment, as in the lost child usage scenario above, is impossible with an install-based deployment model. Users also have to be tolerant of the processing, memory, and battery life these applications consume. Because today's mobile operating systems are designed to shield applications from each other, each application is meant to be self-contained and does not share information with others. In addition, some sensors, such as cameras, need to be exclusively locked before use. Participating in more than one crowdsensing application at a time is therefore not easy, even if a user is positively inclined.

The third obstacle, which primarily affects future applications, is the increasing network bandwidth demands of emerging crowdsensing applications. Table 1 shows that the GPS data has been the most widely used sensing information in the existing crowdsensing applications. However, looking ahead, we envision growing use of data-rich, multimedia sensing information like video [1] in emerging applications such as augmented reality, or the video-based lost child locator discussed above. These applications not only demand far more computing power, but also far more network bandwidth to send data to the cloud infrastructure. Based on data rate analysis of 80 videos on YouTube captured from a first-person viewpoint, each participant in a video-based crowdsensing application will upload between 0.6 Mbps (360p resolution) and 5.6 Mbps (1080p resolution). With many users, such an application can easily overwhelm link capacity in regional networks and into datacenters. For example, Verizon recently introduced state-of-the-art 100 Gbps links in their metro networks [21], yet these are only capable of supporting 1080p streams from just 18000 users. A broadly-deployed application with 1 million users will require 1–2 Tbps, 200x the total upload bandwidth of all YouTube contributors [33] today. An application model where each device sends data to centralized servers (as is typical today) cannot scale to support data-rich sensors. Ensuring the scalability of crowdsensing with data-rich sensors requires rethinking application and cloud architectures to acquire, process, and aggregate such data efficiently.

Ultimately, all three of these obstacles are ramifications of the deployment model in vogue today, where participation in each crowdsensing activity requires a separate application

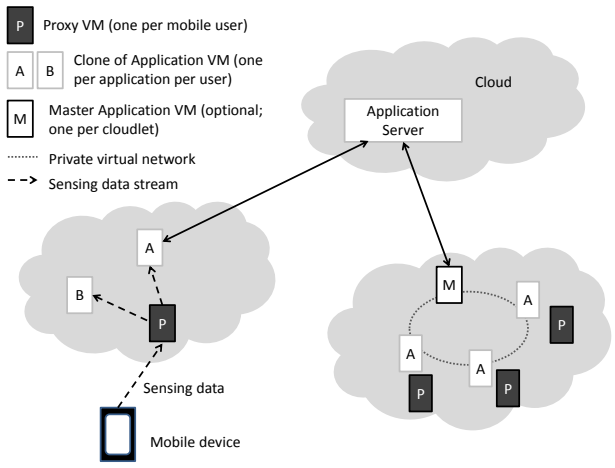


Figure 1: System Architecture.

that must be installed and run on user devices, and directly communicates to central servers. To overcome these obstacles, we must rethink the structure and deployment model used in crowdsensing applications.

3. PROPOSED SOLUTION

We propose a crowdsensing deployment model built around 3 core design principles:

- separation of data collection and sharing from application-specific logic.
- removal of application installation on smartphones from the critical path of application deployment.
- decentralization of processing, and data aggregation near the source of data.

These design principles address the obstacles discussed above. By construction, our proposed solution overcomes the key barriers to scaling up crowdsensing applications.

3.1 System Architecture

The 3-tier system architecture of our deployment model is illustrated in Fig. 1. The first layer is composed of mobile devices, whose roles are essentially reduced to that of (multi-input) sensors forwarding captured data to *proxy VMs* in the second layer. The second layer comprises of distributed cloud infrastructure deployed close to the mobile devices, typically in the access or aggregation network of Wi-Fi or cellular network providers. The concept of distributed cloud infrastructure here is akin to the concept of *cloudlet* presented in [26]. In practice, this can be a private cloud owned by a business or community, or a small data center such as Myoonet's Micro data center [18] that is deployed by a cloud operator. For the sake of simplicity, we will refer to this distributed cloud infrastructure as cloudlets in the remainder of this paper.

Each proxy VM is associated with a single mobile device, and is kept physically close to the mobile device through VM migration to other cloudlets or public clouds. This ensures network resources to transfer data from the mobile device is minimized. The proxy VM handles all the requests for sensor data on behalf of the mobile device. On the mobile device, a single application is responsible for collecting sensor data and communicating it to the proxy VM. This application can be either implemented as a native application, or -if a good mobile browser is available on the device- as a HTML5 web application. The proxy VM is essentially an extension of the mobile device into the cloudlet, and can perform custom data preprocessing, e.g., to enforce privacy settings or handle quirks of the mobile platform, and enforce user preferences on data sharing and crowd participation. From here, data is forwarded to one or more *application VMs* also running on the cloudlet infrastructure.

Application VMs perform data processing steps specific to each crowdsensing application. Each application VM hosts a single crowdsensing application, which is not customized to any particular mobile platform. Generally, for each crowdsensing activity, one application VM is assigned to each participant, making it easy to migrate a user's proxy VM together with the associated application VMs, preserving any hard state they may contain. If an application does not need to maintain hard state for each user, then a single application VM can be shared by all users on a particular cloudlet.

The application VMs for each sensing service are deployed by a coordinating entity on the highest layer in our architecture, typically by the application server running on the centralized cloud infrastructure. In practice, when many application VMs are run on each cloudlet, the application server can initiate a master application VM (MAVM) on each cloudlet and delegate management and data aggregation tasks. The MAVM will coordinate, clone, and configure the application VMs on the cloudlet, and aggregate data within the cloudlet before forwarding results. Depending on the application, the MAVMs on multiple cloudlets may form a peer-to-peer overlay network / tree to scalably aggregate data to the central application server.

Our deployment model is predicated on two assumptions. First, this architecture depends on distributed cloud infrastructure near the user. The vision of executing customized VMs on nearby infrastructure has been articulated many times, e.g. in [8] and [11]. It has also be argued that offloading to nearby computing infrastructure (cyber foraging) is needed for compute-intensive and latency-sensitive mobile applications [4] for the purpose of energy savings and latency reduction. Our concept of distributed cloud infrastructure to host proxy and application VMs fits perfectly in this vision.

Second, our approach assumes a standard API exists for the data transfer between the proxy VM and the associated application VMs. However, we argue this is a much easier task to accomplish than having to write an individual application for each mobile platform (and possibly for each individual version of the mobile platform). Indeed, the output of scalar sensors can be represented as a few integers (e.g. GPS coordinates, temperature value, ...), and standards for multimedia data (e.g., video formats) already exist. Combining such data with standardized XML format descriptions, one can establish a standard for communication between proxy VMs and application VMs. In fact, several programming frameworks for crowdsensing applications have proposed solutions to abstract sensing information [32, 24] and task description [22]. These programming frameworks can be leveraged in our model as well.

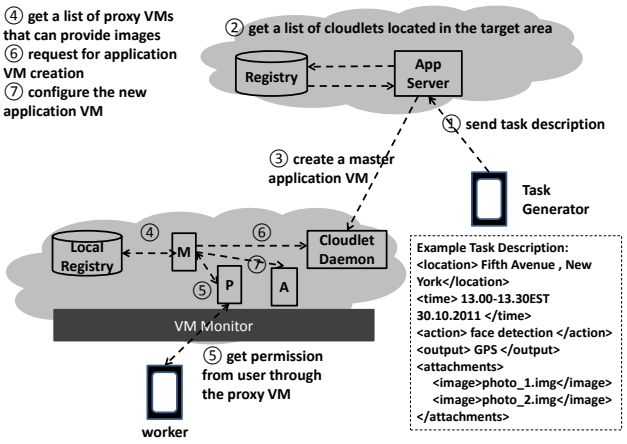


Figure 2: Workflow of crowd bootstrap.

3.2 Crowd Bootstrap

Let's revisit the lost child scenario from Section 1 to see how a crowdsensing task can be rapidly bootstrapped using our deployment model. As shown in Fig. 2, the process of crowd bootstrap can be summarized in the following seven steps.

1. The task generator (here, it is the parents' smartphone) constructs and sends a task description to the application server, typically located in the public cloud. The actual format of this can be application-specific, but is shown as an XML snippet here. The critical information includes type of search (face detection), the sample images, and a location area to scope the search. How this description is constructed and communicated is also left to the application, e.g., with a front-end app on the phone, or through a web-form on the application server.

2. The application server parses the task description, and consults a global registry for a list of cloudlets that are located in the target area.

3. The application server contacts the cloudlet daemon on each target-area cloudlet, and requests a MAVM instance be created. It forwards the VM disk image and memory snapshot to launch the MAVM. In practice, techniques employing demand paging or VM synthesis can minimize overheads of launching the MAVM.

4. The MAVM on each target cloudlet uses a cloudlet-local registry to discover proxy VMs connected to devices that can provide the desired sensor data (here, videos and images).

5. The MAVM requests participation from the mobile users through the proxy VMs. Depending on user-defined policies, the proxy may require explicit permission from the user, or the proxy VM can automatically join crowds on behalf of the user when particular criteria are met (e.g., share video when in a public space, but not audio).

6. Once permission is granted, the MAVM will request the cloudlet daemon to create application VMs. In practice, these can simply be clones of the MAVM, operating in a different mode.

7. The MAVM configures the networking setup of the application VMs, while the the proxy VM will add the new application VM to the subscriber list.

When the above steps are finished, the proxy VMs will start forwarding images and videos to the application VMs, which will apply face detection and forward potential matches through the MAVM and application server to the parents' smartphone. We believe our architecture has the potential to bootstrap large crowds in just a matter of minutes, making this on-demand crowdsensing use case possible.

3.3 Benefits of Our Design

Our deployment model is architected to support scalable, efficient data sharing between multiple applications and users, while reducing the burden on application developers and end users. It scales up crowdsensing tasks by making it easier to access data from a larger pool of diverse smartphones, allow users to simultaneously particpate in multiple applications, and support rich, high-data-rate sensors at global scale.

Separating the process of data collection and sharing from application-specific processing, our system lets developers focus on the latter, rather than porting their application to a myriad of mobile platforms and understanding the idiosynchrasies of each. In fact, our deployment model increases the choices in programming languages, as the application is self-contained in its application VM and does not have to meet specific compatibility constraints for mobile platforms. Similarly, the developer is free to use a variety of programming models to distribute computation and aggregate results, and not forced to use a one-size-fits-all paradigm. Deploying VMs to users boils down to rapid cloning of the application VMs on cloudlets, regardless of the mobile devices.

In our architecture the personal data of the users is stored and processed on their own proxy VMs. According to [7] this approach provides a higher degree of privacy if compared to the traditional approach of storing and processing the data using centralized third party services. Our framework also allows flexibility in partitioning work between the proxy VM and mobile device. For example, supporting multiple applications with differing fidelity or resolution requirements simultaneously will entail some amount of preprocessing; this can be done in the proxy VM, mobile device, or a combination of both depending on hardware capabilities, processing overheads, and energy availability.

Our approach reduces the burden on users and their mobile devices for participating crowdsensing. First, instead of installing individual apps on their devices for each crowdsensing application, users only need to install one app that allows users to participate in different crowdsensing applications. Users can join a crowd by simply granting permission, and if willing, can direct their proxies to automatically participate in some forms of crowdsensing. When a user leaves a crowd, the application VM is simply destroyed, and does not require additional attention from the user. Second, demands on the mobile device can also be reduced, as processing is offloaded to the cloud, and only a single copy of the sensor data is uploaded even when participating in multiple applications. The potential reduction in data transmission helps save energy for users' mobile devices.

Lastly, our design performs processing and data aggregation close to the data sources. This brings two benefits: 1) it reduces traffic on wide-area networks; 2) it reduces network latency by avoiding long-distance data transmission through the backbone networks. This makes it possible to scale up crowdsensing with high-data-rate sensors. VM migration can ensure that processing remains close to data source even as users move around.

4. CHALLENGES

There are several technical hurdles in the path of a real-world deployment of our proposed architecture. We discuss these below.

4.1 Virtualization Overhead

Leveraging virtualization allows us to create a flexible platform in a multi-party setting where user privacy, scalability and isolation between crowdsensing applications are key requirements. These advantages come at the price of both VM creation overhead and the need for more advanced inter-VM communication management. In our design, a new clone of the application VM is instantiated for each user joining the crowd. Ideally, this new VM should start as fast as possible with minimal cost on resources. In practice, when a VM Monitor starts a new VM, it must first reserve all of the memory resources needed for the VM. This constraint prevents rapid creation of multiple VMs concurrently.

One way to solve this problem is to reduce the number of running VMs by replacing the per-user application VMs with one multiplexing application VM on each cloudlet. However, this will introduce the complexity of process migration in mobile scenario when any hard state contained in the application VMs must be preserved. An alternative way is to reduce the overhead of VM creation through advanced cloning mechanisms. There are several efforts that try to reduce the memory copy overhead by cloning the memory from running VMs. SnowFlock [13] proposes to fetch memory on demand while cloning VMs. It manages to clone 32 clones in 32 different hosts within one second by combining on-demand fetching with TCP multicasting for network scalability. Kaleidoscope [6] takes this one step further by discriminating VM memory state into semantically related regions to achieve prefetching and efficient transmitting.

An additional challenge is configuration and performance of inter-VM communication. The performance of inter-VM communication is relatively low compared to inter-process communication. When the system workload on the cloudlet increases, this may result in delayed transmission of sensor data between proxy and application VMs. Note that this low performance is due to inefficient CPU scheduling of the host, as the physical network interface is not touched by inter-VM traffic.

4.2 Migration-induced Reconfiguration

Physical mobility of a device may trigger the migration of the proxy VM and the associated application VMs that are not stateless. Consequently, the IP address of the mobile device as well as those of the VMs may change. To maintain established connections between mobile device and proxy VM, as well as between proxy VMs and application VMs, automated advanced network reconfiguration is needed. This potentially includes network addressing, NAT settings and firewall setup in VMs. Due to this overhead, IP-based solutions may not provide adequate performance in our envisaged scenarios. Non IP-based solutions such as the Host Identify Protocol [19] have been designed from scratch with these limitations in mind, but these protocols still need to be evaluated in real networks. The deployment of these is unlikely to be easy, given the fact that today's Internet is built almost exclusively on the TCP/IP stack.

4.3 Standardization of Sensing Interfaces

Sensor data is distributed from the proxy VMs to the application VMs through a publish-subscribe mechanism. Standard sensor data descriptions are needed to realize communication between proxy VMs and application VMs of various developers. As discussed in Section 3.1, some efforts [32, 24] have been invested on developing such interfaces, however, unfortunately so far no consensus has been made yet.

Another challenge lies in the fact that different crowdsensing applications might be built on the same sensor data, but require a different format or sample rate. However, the sensor data collected from the devices provided by different vendors may not be able to always provide the data in the right format or at the right sample rate.

There is a trade-off to be studied on whether the conversion from the original sensor data to the requested output format(s) must be done on the mobile device, the proxy VM or inside the application VM itself. At first sight, running inside the application VM is the most logical choice, as it removes as much logic as possible from the mobile device and the proxy VM. However, this results in a lack of synchronization and a potential waste of resources. For example, what if all currently running application VMs only need camera frames at 10 fps, while the mobile device emits at a standard 30 fps? In this case, it would make sense to put downsampling application logic on the mobile device, and to put logic in the proxy VM that can configure the sensor capturing on the mobile device. When a new application VM is deployed needing 15 fps, the proxy VM may instruct the mobile device to increase its frame rate accordingly. Support for device-level configuration may vary significantly by platform and specific sensor hardware, so proxy VMs need to be designed to abstract away such differences.

5. CONCLUSIONS

This paper has argued that the existing deployment model for crowdsensing applications does not support either efficient crowd scaling over heterogeneous mobile platforms or the data sharing between crowdsensing applications. While VM-based cloudlets have been widely studied and utilized for computation offloading, we explore the potential uses of VM-based cloudlets for lowering the barriers to scaling up crowdsensing applications. Our solution leverages the existing programming frameworks for crowdsensing applications. There are still several challenges that must be addressed before this kind of deployment model can be adopted, we are implementing the deployment platform with specific focus on the research challenges discussed in this paper.

6. ACKNOWLEDGMENTS

We are grateful to the anonymous reviewers and our shepherd Nina Bhatti for their valuable comments and feedback, which helped to improve the final version. This research was supported by the National Science Foundation (NSF) under grant numbers CNS-0833882 and IIS-1065336, by an

Intel Science and Technology Center grant, by the Department of Defense (DoD) under Contract No. FA8721-05-C-0003 for the operation of the Software Engineering Institute (SEI), a federally funded research and development center, and by the Academy of Finland under grant number 253860. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily represent the views of the NSF, Intel, DoD, SEI, Carnegie Mellon University or Academy of Finland. This material has been approved for public release and unlimited distribution except as restricted by copyright.

7. REFERENCES

[1] Bahl, P., Philipose, M., and Zhong, L. Vision: cloud-powered sight for all: showing the cloud what you see. In *Proc. of MCS* (2012).

[2] Balan, R. K., Nguyen, K. X., and Jiang, L. Real-time trip information service for a large taxi fleet. In *Proc. of MobiSys* (2011).

[3] Bao, X., and Roy Choudhury, R. Movi: mobile phone based video highlights via collaborative sensing. In *Proc. of MobiSys* (2010).

[4] Bonomi, F., Milito, R., Zhu, J., and Addepalli, S. Fog computing and its role in the internet of things. In *Proc of MCC* (2012), pp. 13–16.

[5] Bozzon, A., Brambilla, M., and Ceri, S. Answering search queries with crowdsearcher. In *Proc. of WWW* (2012).

[6] Bryant, R., Tumanov, A., Irzak, O., Scannell, A., Joshi, K., Hiltunen, M., Lagar-Cavilla, A., and de Lara, E. Kaleidoscope: cloud micro-elasticity via vm state coloring. In *Proc. of EuroSys* (2011).

[7] Cáceres, R., Cox, L., Lim, H., Shakimov, A., and Varshavsky, A. Virtual individual servers as privacy-preserving proxies for mobile devices. In *Proc of MobiHeld* (2009), pp. 37–42.

[8] Cuervo, E., Balasubramanian, A., Cho, D.-k., Wolman, A., Saroiu, S., Chandra, R., and Bahl, P. Maui: making smartphones last longer with code offload. In *Proc. of MobiSys* (2010).

[9] Fujisaka, T., Lee, R., and Sumiya, K. Discovery of user behavior patterns from geo-tagged micro-blogs. In *Proc. of ICUIMC* (2010).

[10] Ganti, R. K., Ye, F., and Lei, H. Mobile crowdsensing: current state and future challenges. *IEEE Communications Magazine 49*, 11 (2011).

[11] Kosta, S., Aucinas, A., Hui, P., Mortier, R., and Zhang, X. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *Proc of INFOCOM* (2012).

[12] Koukoumidis, E., Peh, L.-S., and Martonosi, M. R. Signalguru: leveraging mobile phones for collaborative traffic signal schedule advisory. In *Proc. of MobiSys* (2011).

[13] Lagar-Cavilla, H. A., Whitney, J. A., Scannell, A. M., Patchin, P., Rumble, S. M., de Lara, E., Brudno, M., and Satyanarayanan, M. Snowflock: rapid virtual machine cloning for cloud computing. In *Proc. of EuroSys* (2009).

[14] Lane, N. D., Xu, Y., Lu, H., Hu, S., Choudhury, T., Campbell, A. T., and Zhao, F. Enabling large-scale human activity inference on smartphones using community similarity networks (csn). In *Proc. of UbiComp* (2011).

[15] Lu, H., Pan, W., Lane, N. D., Choudhury, T., and Campbell, A. T. Soundsense: scalable sound sensing for people-centric applications on mobile phones. In *Proc. of MobiSys* (2009).

[16] Mathur, S., Jin, T., Kasturirangan, N., Chandrasekaran, J., Xue, W., Gruteser, M., and Trappe, W. Parknet: drive-by sensing of road-side parking statistics. In *Proc. of MobiSys* (2010).

[17] Mobile HTML5. `http://mobilehtml5.org`, 2013.

[18] MYOONET. Unique scalable data centers. `http://www.myoonet.com/unique.html`, 2011.

[19] Nikander, P., Gurtov, A., and Henderson, T. Host identity protocol (hip): Connectivity, mobility, multi-homing, security, and privacy over ipv4 and ipv6 networks. *Communications Surveys Tutorials, IEEE 12*, 2 (2010), 186 –204.

[20] Niu, Z., Li, S., and Pousaeid, N. Road extraction using smart phones gps. In *Proc. of COM.geo* (2011).

[21] PC World. `http://www.pcworld.com/article/255519/verizon_to_offer_100g_links_resilient_mesh_on_optical_networks.html`, 2012.

[22] Ra, M.-R., Liu, B., La Porta, T. F., and Govindan, R. Medusa: a programming framework for crowd-sensing applications. In *Proc. of MobiSys* (2012).

[23] Rabbi, M., Ali, S., Choudhury, T., and Berke, E. Passive and in-situ assessment of mental and physical well-being using mobile sensors. In *Proc. of UbiComp* (2011).

[24] Ravindranath, L., Thiagarajan, A., Balakrishnan, H., and Madden, S. Code in the air: simplifying sensing and coordination tasks on smartphones. In *Proc. of HotMobile* (2012).

[25] Satyanarayanan, M. Mobile computing: the next decade. In *Proc of MCS* (July 2010).

[26] Satyanarayanan, M., Bahl, P., Caceres, R., and Davies, N. The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Computing 8*, 4 (Oct. 2009).

[27] Shah, S., Bao, F., Lu, C.-T., and Chen, I.-R. Crowdsafe: crowd sourcing of crime incidents and safe routing on mobile devices. In *Proc. of ACM SIGSPATIAL GIS* (2011).

[28] Tang, A., and Boring, S. #epicplay: crowd-sourcing sports video highlights. In *Proc. of CHI* (2012).

[29] Wakamiya, S., Lee, R., and Sumiya, K. Crowd-sourced urban life monitoring: urban area characterization based crowd behavioral patterns from twitter. In *Proc. of ICUIMC* (2012).

[30] Wirz, M., Strohrmann, C., Patscheider, R., Hilti, F., Gahr, B., Hess, F., Roggen, D., and Tröster, G. Real-time detection and recommendation of thermal spots by sensing collective behaviors in paragliding. In *Proc. of SCI* (2011).

[31] Yan, T., Kumar, V., and Ganesan, D. Crowdsearch: exploiting crowds for accurate real-time image search on mobile phones. In *Proc. of MobiSys* (2010).

[32] Ye, F., Ganti, R., Dimaghani, R., Grueneberg, K., and Calo, S. Meca: mobile edge capture and analysis middleware for social sensing applications. In *Proc. of WWW Companion* (2012).

[33] YouTube. `http://www.youtube.com/t/press_statistics`, 2012.

[34] Zheng, Y., Zhang, L., Xie, X., and Ma, W.-Y. Mining interesting locations and travel sequences from gps trajectories. In *Proc. of WWW* (2009).

[35] Zhou, P., Zheng, Y., and Li, M. How long to wait?: predicting bus arrival time with mobile phone based participatory sensing. In *Proc. of MobiSys* (2012).

[36] Zimmerman, J., Tomasic, A., Garrod, C., Yoo, D., Hiruncharoenvate, C., Aziz, R., Thiruvengadam, N. R., Huang, Y., and Steinfeld, A. Field trial of tiramisu: crowd-sourcing bus arrival times to spur co-design. In *Proc. of CHI* (2011).

Open Data Kit 2.0: Expanding and Refining Information Services for Developing Regions

Waylon Brunette, Mitchell Sundt, Nicola Dell, Rohit Chaudhri, Nathan Breit, Gaetano Borriello
Department of Computer Science and Engineering
University of Washington
Box 35350, Seattle WA, 98195, USA
{wrb, msundt, nixdell, rohitc, nbnate, gaetano}@cse.uw.edu

ABSTRACT
Open Data Kit (ODK) is an open-source, modular toolkit that enables organizations to build application-specific information services for use in resource-constrained environments. ODK is one of the leading data collection solutions available and has been deployed by a wide variety of organizations in dozens of countries around the world. This paper discusses how recent feedback from users and developers led us to redesign the ODK system architecture. Specifically, the design principles for ODK 2.0 focus on: 1) favoring runtime languages over compile time languages to make customizations easier for individuals with limited programming experience; 2) implementing basic data structures as single rows within a table of data; 3) storing that data in a database that is accessible across applications and client devices; and 4) increasing the diversity of input types by enabling new data input methods from sensors. We discuss how these principles have led to the refinement of the existing ODK tools, and the creation of several new tools that aim to improve the toolkit, expand its range of applications, and make it more customizable by users.

Categories and Subject Descriptors
H.4 Information Systems Applications

General Terms
Design

Keywords
Open Data Kit, mobile computing, smartphones, ICTD, sensing, mobile databases, spreadsheets, data tables, paper forms, vision.

1. INTRODUCTION
Smartphones are rapidly becoming the platform of choice for deploying data collection and information services in the developing world. They have quickly leap-frogged desktop and laptop computers due to their mobility, increased independence from the power infrastructure, ability to be connected to the internet via cellular networks, and relatively intuitive user interfaces enabling well-targeted applications for a variety of domains. In effect, developing countries are skipping the desktop and laptop phase of computing development, and are instead using smartphones and tablets for a range of tasks that have traditionally been performed on larger machines. In concert with this development, cloud services are providing many organizations with the ability to easily rent data storage space and scale hosting resources as needed, either locally or anywhere in the world.

We recognized two trends - 1) capable client devices with rich user interfaces and 2) cloud-based scalable data collection, computing, and visualization services - several years ago when we began the Open Data Kit (ODK) project at the University of Washington. Through ODK, we sought to create an evolvable, modular toolkit for organizations with limited financial and technical resources to use to create data collection and dissemination services. We chose Android as our development platform because its flexible inter-process communication methods allowed us to use existing apps for taking pictures, scanning barcodes, and determining location, rather than having to rewrite them ourselves, thus speeding development. ODK's development was guided by a few simple principles, namely:

- Modularity: create composable components that could be easily mixed and matched, and used separately, or together;
- Interoperability: encourage the use of standard file formats to support easy customization and connection to other tools;
- Community: foster the building of an open source community that would continue to contribute experiences and code to expand and refine the software;
- Realism: deal with the realities of infrastructure and connectivity in the developing world and always support asynchronous operation and multiple modes of data transfer;
- Rich user interfaces: focus on minimizing user training and supporting rich data types like GPS coordinates and photos;
- Follow technology trends: use consumer devices to take advantage of multiple suppliers, falling device costs, and a growing pool of software developers.

The name ODK refers to the entire suite of modular tools. Each tool in the suite has been assigned a name that describes its function. Previous work has discussed ODK 1.0 [6], which consisted of three primary tools: Build, Collect, and Aggregate. These provide the ability to design forms, collect data on mobile devices (e.g. phones, tablets), and organize data into a persistent store where it can be analyzed. Prior papers have also described the design of several new tools that are being incorporated into the ODK suite: Sensors [1], Scan [3] and Tables [7]. As we deployed the original tools, gathered feedback from users, and sought to incorporate new tools into the ODK suite, it became clear that there were some deficiencies in our design that needed

ACM HotMobile'13, February 26-27, 2013, Jekyll Island, Georgia, USA.

to be addressed. This paper describes these deficiencies, and the rationale that drove a redesign of the inter-tool architecture of the tool suite, which will be released under the ODK 2.0 label.

ODK has quickly become one of the leading solutions for a wide variety of organizations, from small NGOs to large government ministries, and now has thousands of users in dozens of countries around the world. The projects for which it is being used range over an ever-increasing set of domains including public health (our original focus), environmental monitoring, and documenting human rights abuses. The ODK website has been visited by over 65,000 unique visitors from 202 different countries/territories and averages over 8500 hits a month. Additionally, over 11,000 distinct users have installed Collect from Google Play (a number that does not include organizations that install Collect directly when setting up their deployment). From our users' collective experience using ODK, we have seen many ways to improve the toolkit, expand its range of applications, and make it even more customizable. Recently, we conducted an extensive survey of the ODK user and developer community to better understand how people are using ODK and how organizations' data collection needs are evolving. 73 organizations completed our survey, providing information on 55 different deployments involving at least 5500 mobile devices in over 30 countries. This vast amount of feedback, in conjunction with the numerous deployment reports and feature requests submitted to our mailing list and website, led us to rethink the ODK system architecture. This paper reports on the changes that we are now implementing to our system architecture and applications, and the rationale behind each.

2. LIMITATIONS OF ODK 1.0

Our observations and survey responses can be grouped into four principal areas of refinement for ODK:

1. support data aggregation, cleansing, and analysis/visualization functions directly on the mobile device by allowing users to view and edit collected data;
2. increase the ability to change the presentation of the applications and data so that the app can be easily specialized to different situations without requiring recompilation;
3. expand the types of information that can be collected from sensing devices, while maintaining usability by non-IT professionals; and
4. incorporate cheaper technologies such as paper and SMS into the data collection pipeline.

The design of ODK 1.0 focused on collecting data in the form of surveys, and uploading completed surveys into the database for analysis and aggregation. It did not provide facilities for getting that data back out to clients to review and update. However, feedback shows that many users want to be able to store already collected data (either from past data collection or from a server database) on the device and use it to specify which data to display (e.g., a patient's past blood pressure readings) or to steer survey logic (e.g., select follow-up questions based upon a patient's medical history). One user told us, *[One limitation of ODK 1.0] is the lack of a local database on the device [that contains] previously collected information. For example, the last time I visited your household, there were 5 people living here. Are those 5 people still living here?* In addition, one of the largest requests that we received from users is to make it possible to view and edit data on the device. For example, one user told us, *We need a presentable way of viewing collected data on the device … like if you have a roster and need to make decisions based on some earlier responses, you need to be able to view this data.*

Rendering of the surveys in ODK 1.0 was accomplished using a variant of a W3C XForms standard defined by the OpenRosa Consortium. Although XForms can specify input constraints (to provide some immediate error checking abilities), form navigation logic (branching based on previous answers), and multiple languages (for local customization), It does not describe the visual presentation of the prompts and data types. This led to many specializations of ODK for different organizations, which was technically challenging for many users. One user told us, *We struggled to understand xml and the XForm. While the XForm is fairly simple, the xml structure is confusing. Some of the advanced features require core knowledge of xml coding.*

Originally, ODK assumed humans would enter data explicitly or, at most, gather data from sensors that were built-in to the device (such as GPS coordinates, barcodes, photos, audio, and video). It did not support the ability to interact with new sensors or process data captured from built-in sensors like the camera. However, gathering information from external sensors is an often-requested feature; such requests range from enhancing a health survey with data obtained from medical sensors, to automatically incorporating GPS and compass data with captured photos. One user told us, *Our [use case] requires us to measure the height of trees. We currently use a clinometer for this and enter the data manually. It would be great if we could access the clinometer [from the device] and use it as part of our data collection process*. Collecting data from sensors attached to the mobile device is attractive because applications can directly receive and process the data, obviating the need for manual data transfer by a human, which may be error-prone.

Finally, many organizations have extremely limited financial resources and still rely on paper forms or very cheap mobile phones to gather data, and there is a need to connect these media to the ODK ecosystem. Simple SMS is a very common form of communication on mobile devices, particularly in developing countries, where many people use basic mobile phones that have only text and voice features. In addition, many organizations are unable to afford the cost of purchasing and maintaining a mobile device for every field worker. Such organizations would prefer to use cheap and well-understood paper forms to collect data at the lowest level of the information hierarchy, and then digitize the data at a higher level to enable data transmission, statistical analysis, and aggregation. The limitations of the original ODK 1.0 tools are addressed by the design of ODK 2.0, a refined and expanded toolkit with a more flexible system architecture.

3. DESIGN OF ODK 2.0

The refinement and expansion of ODK is based on four core design principles that we are incorporating into all the tools (these stem directly from the four areas of refinement described at the beginning of Section 2 but do not correspond 1-to-1):

1. when possible, UI elements should be designed using a more widely understood runtime language instead of a compile time language, thereby making it easier for individuals with limited programming experience to make customizations;
2. the basic data structures should be easily expressible in a single row, and nested structures should be avoided when data is in display, transmission, or storage states;
3. data should be stored in a database that can be shared across devices and can be easily extractable to a variety of common data formats; and

4. new sensors, data input methods and data types should be easy to incorporate into the data collection pipeline by individuals with limited technical experience.

Our evolved system architecture is still governed by the over-arching concern that for computing tools to address the many information gaps in developing regions, information services must be composable by non-programmers and be deployable by resource-constrained organizations (in terms of both financial and technical resource constraints) using primarily consumer services and devices. To facilitate this, the new ODK 2.0 toolkit (individual tool names are italicized) provides a way to synchronize, store and manipulate data in *Tables* on a mobile device with a user interface that supports both the smaller smartphone screen and larger tablet form-factors, and allows viewing and manipulating data in a simple row format. In addition, the new design makes customization easier by using widely understood standard presentation languages, such as HTML and JavaScript, to facilitate a more easily tailored user experience on a per *Survey* basis. Furthermore, ODK 2.0 makes is possible to attach external *Sensors* to mobile devices over both wired and wireless communication channels, thereby reducing the amount of manual data transcription from sensors into survey forms, and also facilitates the automatic conversion of information recorded on paper forms to a digital format by using the camera of the mobile device to *Scan* documents. See Figure 1 for a block diagram of the ODK tool suite architecture.

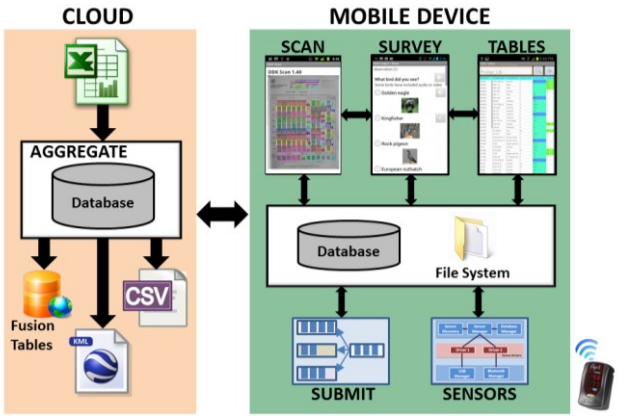


Figure 1: The new ODK 2.0 system architecture, showing cloud services (left) and mobile client services (right). In the cloud, Aggregate provides services to synchronize data across devices and export data in common file formats. On the device, a common database/file system is shared between the tools and across clients. Scan, Survey, and Tables (above the database) are tools for gathering, processing and visualizing input data; Submit and Sensors (below the database) are tools that augment Android to create additional services.

3.1 Data Management on Mobile Device

Many applications rely on previously collected data; for example, logistics management, public health, and environment monitoring often require workers to return and reference previously collected data to verify and possibly update conditions. In the previous ODK design, revising data from previously completed surveys was not supported. However, more and more of our users want to be able to use all or part of previous surveys to complete new ones (e.g., not re-entering patient demographics for a follow-up visit when that data was already collected in the original registration form). To enable data updating, aggregation, curation, cleansing and analysis functions on the mobile platform we created Tables. Tables allows a user to create new tables, add data, delete data, search data, scroll through data, add columns, configure data types, view graphs, apply conditional formatting, perform summary calculations, and synchronize the data with the cloud. Tables presents a user interface for editing and viewing data that is optimized for the smaller screens of mobile devices [7], and provides customization capabilities for users to easily configure the app for their use case. Tables has a number of built-in views, and allows users to explore their tabular data with customizable views defined by HTML/JavaScript files, thus making presentation much more flexible while avoiding recompilation. These views can pull data from, and link to, other tables, so that users can form an integrated app, rather than a set of loosely connected (or completely disconnected) tables. For example, a table of facilities can link to a table of specifics about individual resources at each facility. Alternatively, Tables can use Survey's strong data typing to add and edit entire rows and incorporate input constraint checking, or use Scan's image processing capabilities to add rows based upon filled-in paper forms. These tools (Tables, Survey, and Scan) use an inter-tool architecture based on a common SQLite database schema.

Another important refinement to Survey is making data collection and presentation more easily customizable. In Collect, changing the look-and-feel of a particular question type or extending the expression language (e.g., adding a count function) to express the user's business logic (e.g., visibility and value constraints) required changes to Java source code. This high barrier to change meant that we spent a significant amount of time refining our user interface because it needed to be generic enough to work for many use cases. It also created friction to the adoption of the technology because organizations lacked the skills or funding necessary to customize the tool. In contrast, Survey allows organizations to easily express their business logic and heavily customize the user interface for their specific use case through the use of JavaScript and HTML. We anticipate that Survey's JavaScript form interpreter, the use of open source toolkits (e.g., JQueryMobile, Handlebars, Backbone), and the greater worldwide prevalence of JavaScript and HTML coding skills will make it easier for individuals and organizations to make domain-specific customizations. Our design leverages the suite of standard ODK question widgets that encapsulate the rendering, event handling and business logic. These question widgets are then extended at runtime to incorporate rendering and business logic customizations (e.g., visibility and value constraints). Users can easily customize the user interface by specifying an alternative Handlebars template in the form definition, causing the widget to render using the alternative template.

User experiences from ODK 1.0 deployments show that although non-technical users are able to make small customizations to existing XForms, creating an entire XForm from scratch is often too challenging. Thus, to shield users from the complexity of writing XForms, we created Build, a tool that allows users to graphically compose surveys, and XLSForm (based on 'pyxforms' [14]) that gives users the option of writing their survey in an Excel spreadsheet that is automatically converted to an XForm. In ODK 2.0, we are building a revised converter to transform a spreadsheet to a JSON description that can be rendered using ODK's new interpreter that leverages web technologies. By allowing users to specify information in a spreadsheet, it enables non-technical users to remain shielded from the complexity of writing JSON and JavaScript. Users with minimal Javascript and

HTML skills will be able to copy and modify standard template files (e.g., use different HTML constructs or add CSS style classes) and reference these modified template files to customize the rendering of individual questions in the form or create new question types. In the same way, users can also revise the standard templates and CSS stylesheets to create an organization-specific look and feel. Users with more advanced Javascript and HTML skills can customize a question widget's event handling (e.g., add mouseover-like treatments) or define entirely new widgets.

3.2 Improved Input Methods

ODK 2.0 reduces the amount of manual data transcription from sensors into surveys by making it possible to attach external sensors to mobile devices. By hiding complexities such as the management of communication channels and sensor state as well as data buffering and threading, the Sensors framework [1] simplifies the code needed to access a sensor.

Sensors provides a common interface to access both built-in and external sensors connected over a variety of communication channels. Thus far, we have implemented channel managers for Bluetooth and USB, and plan to implement managers for WiFi and NFC in the near future. The USB Manager currently supports three USB protocols: Android's Accessory Development Kit (ADK) 2011, ADK 2012, and a USB Host serial channel. Sensors also provides a convenient built-in sensor discovery mechanism that allows users to discover sensors and associate the appropriate driver with a sensor. Users who want to integrate external sensors with their mobile devices download and install the Sensors app and sensor driver app from an app store such as Google Play. This facilitates the easy delivery of the application and driver updates to devices. Figure 2 (left) shows Sensors being used in a South African clinic to deactivate harmful contaminants (like the HIV virus) in breast milk. Sensors provides abstractions that delineate application code from code that implements drivers for sensor-specific data processing. The sensor driver abstraction allows device drivers to be implemented in user-space so that locked devices can be customized by end users. The framework handles the data buffers and connection state for each sensor, which simplifies the drivers. Separating application code from driver code also allows the code bases to evolve independently.

In addition to accepting and processing input from a variety of different sensors, the continued use of paper forms for data collection in resource-constrained environments made it important that we also facilitate efficient data entry from paper forms. Many of the paper forms used by organizations for data collection contain a mixture of data types, including handwritten text, numbers, checkboxes and multiple choice answers. While some of these data types, such as handwritten text, require a person to manually transcribe the data, others, like checkboxes or bubbles, can be analyzed and interpreted automatically. To take advantage of machine-readability, we designed ODK Scan [3], a piece of software that uses a lightweight JSON form description language to facilitate the processing of existing paper forms without the need to redesign or add coded marks to the forms. To add a form to the system, the user creates a JSON form description file that specifies the size, location and data type of each form field to be processed. The camera on the device is used to photograph the form, and computer vision algorithms use the JSON form description file to automatically segment and interpret the machine-readable data. The image processing components of the application are implemented using OpenCV, an open source computer vision library, while the user interface components are implemented using Android's Java framework. We use the Java Native Interface (JNI) to facilitate communication between the Java framework and OpenCV's native image processing algorithms. All of the image processing is performed on the device so as not to require an Internet or cellular connection. After the image processing is completed, Scan launches Collect so that users can manually complete the entry of data types that are not machine-readable. Scan makes this data entry process faster by exporting small image snippets of each form field to Collect, and the image snippets are displayed on the screen of the device alongside the corresponding data entry box, so that users can simply look at the image snippet and type in the value displayed. Figure 2 (center) shows an image of Scan being used to collect vaccine statistics in a rural health center in Mozambique.

Figure 2: Examples of ODK tools in action. Left: Using Sensors to monitor breast milk pasteurization that deactivates contaminants (e.g. HIV virus); Center: Using Scan to digitize paper based vaccine information in Mozambique; Right: Indigenous tribal member using Collect in the Amazon jungle.

Data can also be collected from and disseminated to users with cheap SMS-only phones. By acting as an SMS server, Tables enables anyone to send SMS messages to query an existing table or add rows to a table. We use the data table abstraction to implement basic access control measures based on the phone number from which the message was sent as well as locally-administered (on the receiving smartphone) usernames and passwords. For example, this allows a farmer with a cheap phone to post available produce to an agent at a remote market or to obtain the commodity prices in that market. This allows Tables to provide services that can be accessed from the cheapest and most common phones without introducing the complexity of an SMS gateway or other cloud-based server.

3.3 Data Management in the Cloud

Less technically capable users encounter significant barriers to leveraging the power of the cloud. To simplify the distribution of forms to mobile devices, the retrieval of data from devices, and storing and managing data, we designed Aggregate, an auto-configuring, ready-to-deploy server. Aggregate manages collected data, provides interfaces to export the aggregated data into standard formats (e.g. CSV, KML, JSON) and allows users to publish data to online services (e.g., Google Spreadsheet or Fusion Tables). Aggregate is a configurable generic data storage service that runs on a user's choice of computing platform (cloud-based or private server). Aggregate can be deployed to the Google AppEngine hosting service to enable a highly-available and

scalable service that can be maintained by unskilled users and less-capable IT organizations. However, many of our users have data locality and security concerns, either because the data cannot legally leave the country of origin, or because the data may contain sensitive identifiable information, or be high-risk or high-value data. For these users, AppEngine may not be appropriate. Aggregate can therefore also run within a Java web container (e.g., Tomcat) using a MySQL or PostgreSQL datastore. Communications security generally relies on HTTPS connections between client devices and the server. However, because many organizations do not have the funds to purchase or the expertise to install SSL certificates on their own servers, we provide user authentication and data security over HTTP communications through DigestAuth and the asymmetric public key encryption of form data before transmission to the cloud. If asymmetric public key encryption is used, the form data is stored in encrypted form on the server, which enables some organizations to continue to leverage the AppEngine cloud hosting service despite stronger data security requirements. In this case, users download the encrypted data to a computer and use a locally-running tool called ODK Briefcase to decrypt it using a private key.

To provide datastore independence, and because Aggregate parses the submitted XForm instance into column values (to better support filtering and visualization) and incorporates a dynamic datastore abstraction layer rather than a layer set at compile-time. Since XForms can define arbitrarily deep nested groupings of repeated questions, Aggregate performs a complex mapping of the XForm to a set of database columns and tables. This greatly complicates the presentation of the data, and the wide variety of different use cases created by users prevents a generic processing of these nested repeating sections when visualizing, publishing or exporting the data. Since Aggregate parses the submitted XForm instance into column values, a more capable data analysis package could be configured to operate directly on the underlying database tables. However, the complexity of this configuration makes it impractical for many of our users.

In ODK 1.0, the communications flow is unidirectional; blank forms flow from the cloud service (Aggregate) to mobile devices, and data from the filled-in forms flows back to the cloud service and then out to remote services or into file exports. Collected data can be deleted, but is otherwise immutable and provides a store of record. Data is stored (aggregated) in the cloud, where simple curation and data visualization tools are provided. Aggregate bridges the gap between mobile data collection tools and the sophisticated data analysis software able to derive complex results by providing many forms of data export.

In version 2.0, a simple row is the basic storage element; repeating groups are explicitly represented as linked rows across two different forms. The new design eliminates the complex backend mapping that made it difficult for organizations to access the database structures directly. The communications flow has changed so it is now a cloud-mediated peer-to-peer store-and-forward network. Any authorized device running Tables can create new surveys and share data with any of its peers and the remote services can publish surveys and data back out to the mobile devices. Retaining a cloud service (Aggregate) as both a datastore and a store-and-forward communications nexus enables robust peer-to-peer operations in intermittent and low-connectivity environments. The cloud also provides a central point from which to manage and disseminate a security model that can be applied and enforced independently on each device.

Since data is no longer immutable, Tables relies upon the user to resolve conflicts that occur whenever two users concurrently update the same row in a table. Conflicts are detected and resolved at the individual row level (in keeping with our row-based information model) between a row on the user's mobile device and a row on the server. This maximizes the system's ability to disseminate new and uncorrelated change across devices. Manual, client-side conflict resolution was chosen because: 1) established recent-modification conflict resolution techniques are inappropriate or difficult to apply across devices that may not be time synchronized and which may be in disconnected operation for extended periods of time; 2) since ODK targets a diverse set of use cases and application domains, any assumptions built into an automatic resolution mechanism will likely be inappropriate for some domains; 3) accurately expressing the procedural rules to be applied during automatic conflict resolution is likely difficult for non-programmers and capturing and applying these domain-specific rules would increase the complexity of the server design; 4) client-side resolution benefits by keeping the user involved with reconciling conflicts since many times they understand the semantics of the conflict and can better resolve it at the moment it is detected rather than by a more remote administrator at a later date.

Data submission is currently initiated by the user because connectivity is often intermittent and organizations want to control data transfer costs. To better use available connectivity that may be sporadic, and to improve data timeliness (both on the mobile device and when publishing data to the peers), we are designing a tool called Submit that will manage data transmission. Submit enables organizations to specify parameters such as data priority, data importance, deadlines, and the cost of the transport mediums. Submit then factors in the device's connectivity history, and intelligently uses the connectivity available (e.g., SMS, GPRS/3G, Wi-Fi) to create a priority routing system that improves data timeliness in the intermittent and expensive connectivity of the developing world. Connectivity history is an important factor in routing decisions, since there may be certain times of day when the device is within range of a Wi-Fi base station. Alternatively, depending on the data priority and the costs of other connectivity options, it may make sense for the data to be stored locally until the user returns to Wi-Fi connectivity.

3.4 Use Case: Cold Chain Management

ODK 2.0 is an expanded and refined set of modular tools for collecting and managing data in low-resource environments. This section describes one concrete use case in which ODK 2.0 could be used to improve the delivery of health and information services. The cold chain is a complex sequence of refrigeration equipment used to ensure that vaccines retain the correct temperature during transport and storage. Collecting and disseminating accurate and timely data regarding a country's cold chain improves resource-allocation and planning, but cold chain inventories are currently mostly paper-based systems that contain large amounts of inaccurate or out-of-date information. Replacing the paper-based system with ODK 2.0 could improve the speed and reliability of the inventory update process. For example, remote field workers could use Tables to automatically download the most up-to-date subset of cold chain data for a site from Aggregate, and use Survey to enter any new refrigerator information. Sensors could be used to continuously monitor the temperatures of refrigerators at the site, and the worker could use Tables to visualize this data and check for anomalies. Finally, the

worker could use Scan to digitize paper-based records that track the number of vaccines administered at this site to improve stock monitoring and resupply. All of these tasks could be performed quickly on-site and the data made immediately available to decision-makers and stakeholders.

4. RELATED WORK

A variety of other solutions attempt to replace paper-based data collection with digital tools. CAM [11] used its own scripting language to augment paper forms by using bar codes to trigger audio prompts for manual data entry. MyExperience [5] collects survey responses triggered by sensor events but does not address the larger issues of organizational information flow. Commcare [4] is the most related to Collect in that it targets use by health workers and also uses XForms, but it is less flexible in how it can be composed with other tools and requires recompilation to customize presentation. Manipulating small databases on phones has received less attention. Tools like Excel are available in smartphone versions but have not been adapted to small screens and do not work directly from a database rather than a file. Uju [13] enables the creation of small databases that can be populated or queried over SMS but does not integrate with tools that obtain data from sensors or paper forms. Extracting data from paper forms via crowdsourcing is being commercialized by Captricity [2], while LocalGround [12] processes manually annotated paper maps and adds the data to existing digital maps. Neither of these tools work in completely disconnected operation. Recent activity focuses on connecting external sensors to phones using audio jacks (Hijack [9]) and Bluetooth (Amarino [8]). Google released a sensor development kit for Android [15], and researchers have focused on low-power operation of external sensors (Reflex [10]). However, what distinguishes ODK 2.0 from other solutions is the interoperability of these elements, and the ability to do all the computation, analysis and visualization on the device.

5. FUTURE WORK & CONCLUSION

The original design of ODK assumed that a system administrator would have access to a computer to initially set up and administer the system, including designing forms and setting up data storage facilities. However, in many rural locations, computers are rarely available, which limits the adoption of ODK in these settings. To reach these areas, it is desirable to create a system that could be entirely set up and administered on a mobile device. While ODK 2.0 provides users with some methods for building information systems on mobile devices (e.g., database design with Tables, customized question widgets) it does not entirely remove reliance on computers as users are not able to configure their cloud service or write a device driver. Additional work is necessary to build a system that can be set up and managed entirely on a mobile device. The new ODK 2.0 design focuses on a core set of tools that enable users to move beyond treating mobile devices as simple input devices, and instead leverage mobile computing platforms to build more dynamic collaborative information systems in the field. We expect that the changes in design and the new capabilities of the software will lead to a rich new set of research challenges and opportunities that we plan to explore.

Open Data Kit provides organizations with a modular toolkit to build application-specific information services for use in resource-constrained environments. Our own experiences combined with extensive feedback from organizations using the toolkit have led to a redesign of ODK that aims to better meet the needs of a wider range of organizations. Specifically, our design changes include 1) favoring runtime languages over compile time languages to make customizations easier for individuals with limited programming experience; 2) implementing basic data structures as single rows, 3) storing data in a database that is accessible across apps and client devices; and 4) increasing the diversity of input types by enabling new data input methods from sensors. We discussed how the new system design led to the addition of several tools to ODK 2.0 and how the new system architecture enables its adaptation to an even larger and varied set of applications. The ODK tools and their source code are freely available for download at http://opendatakit.org and are distributed under an Apache2 license.

6. ACKNOWLEDGMENTS

We gratefully acknowledge the support of Google Research, NSF Grant No. IIS-1111433, and an NSF Graduate Research Fellowship under Grant No. DGE-0718124. We are grateful for all our collaborators at PATH Seattle and the ODK community.

7. REFERENCES

[1] W. Brunette, R. Sodt, R. Chaudhri, M. Goel, M. Falcone, J. Van Orden, G. Borriello, "Open Data Kit Sensors: a Sensor Integration Framework for Android at the Application-level," Proc. 10th Intl. Conf. on Mobile Systems, Applications, & Services (Mobisys), 2012.

[2] K. Chen, A. Kannan, Y. Yano, J. M. Hellerstein, and T. S. Parikh, Shreddr: pipelined paper digitization for low-resource organizations, Proc. 2nd ACM Symp on Computing for Development (DEV), 2012.

[3] N. Dell, N. Breit, T. Chaluco, J. Crawford, and G. Borriello, "Digitizing Paper Forms with Mobile Imaging Technologies," Proc. 2nd ACM Symp on Computing for Development (DEV), 2012.

[4] B. DeRenzi, G. Borriello, J. Jackson, V. S. Kumar, T. S. Parikh, P. Virk, and N. Lesh, "Mobile Phone Tools for Field-Based Health Care Workers in Low-Income Countries," Mount Sinai Journal of Medicine, vol 78, no 3, 2011.

[5] J. Froehlich, M. Y. Chen, S. Consolvo, B. Harrison, and J. A. Landay, "MyExperience: a system for in situ tracing and capturing of user feedback on mobile phones," Proc. 5th Intl .Conf. on Mobile Systems, Applications and Services (Mobisys), 2007.

[6] C. Hartung, A. Lerer, Y. Anokwa, C. Tseng, W. Brunette, and G. Borriello, "Open Data Kit: tools to build information services for developing regions," Proc. 4th ACM/IEEE Intl. Conf. on Information and Communication Technologies and Development (ICTD), 2010.

[7] Y. Hong, H. K. Worden, and G. Borriello, "ODK Tables: data organization and information services on a smartphone," Proc. 5th ACM Workshop on Networked Systems for Developing Regions, 2011.

[8] B. Kaufmann and L. Buechley, "Amarino: a Toolkit for the Rapid Prototyping of Mobile Ubiquitous Computing," Proc. 12th Intl. Conf. on Human Computer Interaction with Mobile Devices and Services, 2010.

[9] Y.S. Kuo, S. Verma, T. Schmid, and P. Dutta, "Hijacking Power and Bandwidth from the Mobile Phone's Audio Interface," Proc. 1st ACM Symp on Computing for Development (DEV), 2010.

[10] F. X. Lin, Z. Wang, R. LiKamWa, and L. Zhong, "Reflex: Using Low-power Processors in Smartphones without Knowing Them," Proc. 17th Intl .Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2012.

[11] T. S. Parikh and E. D. Lazowska, "Designing an Architecture for Delivering Mobile Information Services to the Rural Developing World," Proc. 15th Intl Conf on World Wide Web (WWW), 2006.

[12] S. Van Wart, K. J. Tsai, and T. Parikh, "LocalGround: a Paper-based Toolkit for Documenting Local Geo-spatial Knowledge," Proc. 1st ACM Symposium on Computing for Development (DEV), 2010.

[13] L. Wei-Chih, M. Tierney, J. Chen, F. Kazi, A. Hubard, J. G. Pasquel, L. Subramanian, and B. Rao, "UjU: SMS-based applications made easy," Proc 1st ACM Symp. on Computing for Development (DEV), 2010.

[14] formhub. http://formhub.org/. [Accessed: 11-Oct-2012].

[15] Android Accessory Development Kit. http://developer.android.com/tools/adk/index.html. [Accessed: 11-Oct-2012].

A Framework for Context-Aware Privacy of Sensor Data on Mobile Systems

Supriyo Chakraborty, Kasturi Rangan Raghavan
Matthew P. Johnson, Mani B. Srivastava
University of California, Los Angeles
{supriyo, kasturir, mpjohnson, mbs}@ucla.edu

ABSTRACT

We study the competing goals of utility and privacy as they arise when a user shares personal sensor data with apps on a smartphone. On the one hand, there can be value to the user for sharing data in the form of various personalized services and recommendations; on the other hand, there is the risk of revealing behaviors to the app producers that the user would like to keep private. The current approaches to privacy, usually defined in multi-user settings, rely on anonymization to prevent such sensitive behaviors from being traced back to the user—a strategy which does not apply if user identity is already known, as is the case here.

Instead of protecting identity, we focus on the more general problem of choosing what data to share, in such a way that certain kinds of inferences—i.e., those indicating the user's sensitive behavior—cannot be drawn. The use of inference functions allows us to establish a terminology to unify prior notions of privacy as special cases of this more general problem. We identify several information disclosure regimes, each corresponding to a specific privacy-utility tradeoff, as well as privacy mechanisms designed to realize these tradeoff points. Finally, we propose *ipShield* as a privacy-aware framework which uses current user context together with a model of user behavior to quantify an adversary's knowledge regarding a sensitive inference, and obfuscate data accordingly before sharing. We conclude by describing initial work towards realizing this framework.

Keywords

Behavioral Privacy, Context-awareness, Inferences, Model-based Privacy, Android, ipShield

1. INTRODUCTION

Smartphones with onboard and externally connected body-worn sensors are capable of tracking our locations and social neighborhoods, monitoring physiological markers, and learning about our evolving social dynamics. The raw data collected are increasingly being used to *infer* our personal, social, work and urban contexts. These contexts are in turn acquired by a growing ecosystem of context-aware apps

ACM HotMobile'13, February 26–27, 2013, Jekyll Island, Georgia, USA.

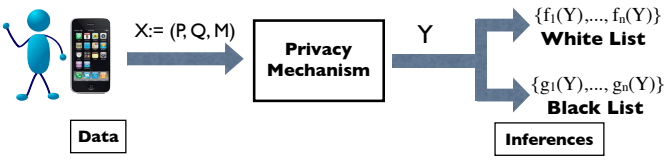


Figure 1: A simplified information flow scenario from users to app producers. The shared data is used for computing various inferences.

to provide us with *personalized* app experiences such as behavior-tailored insurance plans, mobile health (mHealth) diagnostics and customized recommendations to enrich our social and personal interactions (or targeted advertising). We refer to the benefit to the user of such personalization as *utility*. Embedded within the identity-annotated time-series of shared sensor data is the user's full behavioral footprint, to the minutest detail, including many that she may wish to keep private. Users are often unaware of the possible (mis)use of their personal information by the data-consuming untrusted apps, causing information asymmetry between information providers (users) and consumers (apps.), leading to a *lemon market* [25]: users are increasingly skeptical of app producers' privacy policies, with no way of verifying good policies, and so providers have little incentive to abide by them, leading to more user skepticism, in a negative feedback spiral.

Information flow from users to apps is summarized in Fig. 1. Broadly speaking, the shared sensor data X has: (a) a set of personal identifiers P, such as name and SSN associating it to the user; (b) a set of quasi-identifiers Q, such as age, gender, zip code, which when combined with auxiliary information sources can possibly identify the user; and (c) a set M, containing data values corresponding to the measurement. Shared data is represented by Y and is used by the apps to compute various inferences, some of which can be sensitive (i.e., are such that the user wishes them to remain private). Consider the following examples:

E1: A user shares her accelerometer data X with an mHealth app to monitor her overall *activity level* $f(Y)$ but faces the risk of revealing her exact *activity type* $g(Y)$, which is also inferable from the same data.

E2: A user desiring a *safe driver discount* $f(Y)$ on insurance rates may be willing to share her location and accelerometer data X. However, periodic location release near a place of worship may reveal *religious preferences* $g(Y)$.

E3: A user is required to share EKG and respiration data

X with an insurance company which uses the data to check for *heart and respiratory disorders* $f(Y)$, and provide discounted rates. However, the same data can be used to detect *onset of stress* $g(Y)$, a behavior the user wants to keep private.

If Y is the same as X (no privacy mechanism is used), the presence of set P implies that the inferred sensitive behaviors may now be traceable back to the user, violating her privacy.

Prior work on privacy mechanisms is centered around two design objectives: data anonymization and incomplete reconstruction. The process of anonymization includes the removal of P and the suitable *obfuscation* of Q present in X to break the association between the data and the user. In a multi-user setting where privacy of an entire database of user data is desired, measures such as *k-anonymity* [24] and *l-diversity* [17] are used to determine the level of obfuscation required to make the user anonymous or indistinguishable within a subpopulation, achieving privacy-in-numbers. However, the breakdown of anonymization in the face of auxiliary information [18, 11, 23] has prompted the design of measures such as *differential privacy* [6] which, in a multi-user setting, recommend use of structured noise to perturb aggregate query responses and protect the membership (i.e., presence or absence) of an individual within a database. The second objective is to prevent complete reconstruction of X from Y. To achieve this in addition to anonymization, the measurements in M are also adequately perturbed [22]. By preventing reconstruction, the goal is to protect against private inferences which could be made from X alone. However, it has been shown that partially reconstructed data can be used to make inferences about private behaviors [23, 11].

Now consider instead a setting in which a *single* user shares a time-series of sensor data annotated with identity information, as illustrated in $E1 - E3$. This motivates the investigation: *what are the privacy and utility goals appropriate to such a setting?* The traditional notion of protecting user identity is no longer a concern because the apps under consideration (e.g., mHealth, customized insurance plans) require user identity for providing personalized services (utility). Thus, instead of identity, a user is interested in protecting the privacy of sensitive behaviors which can be inferred from the shared data. Another consequence of this single-user setting is that privacy measures relying upon privacy-in-numbers within a subpopulation do not apply.

In this paper, we consider the general form of the privacy problem given above and make three main contributions. First, we give a very general privacy model in which two sets of inferences (the white and black lists shown in Fig. 1) constitute utility and private behaviors, respectively. These inferences are marked as $f(Y)$ and $g(Y)$ in examples $E1 - E3$. The use of inference functions allows us to establish a terminology to unify prior notions of anonymization- and reconstruction-based privacy as special cases of the more general problem. Second, we identify several information disclosure regimes, each corresponding to a specific privacy-utility tradeoff, and privacy mechanisms designed to realize these tradeoff points. These insights lead us to our third contribution: the conceptualization of *ipShield*—a privacy-aware framework which uses current user context together with a model of user behavior to quantify an adversary's knowledge regarding a given sensitive inference, and then apply an appropriate privacy mechanism on the data before sharing. We conclude by describing the initial work we have done towards realization of the framework.

2. PRIVACY PROBLEM AND CHOICES

We define an *inference function* as a classifier which takes shared data as input and performs a classification of the user as being in a particular behavior state, e.g., into one of the activity states (walking, running, still) in $E1$, one of several religions in $E2$, or the onset of stress in $E3$. Classifiers are machine learning algorithms (e.g., supervised, unsupervised, reinforcement) used to learn and then classify based on patterns in the data. For example, in supervised learning, these patterns are learned using labeled data provided by the user.

The problem of protecting the privacy of sensitive inferences is characterized by a tradeoff between the application's need to obtain information for providing *utility* to the user and the user's need to control the information shared for protecting *privacy*. As shown in Fig. 1, our privacy notion is defined in terms of what can be extracted from the shared data Y. The user specifies his privacy preferences as a *blacklist* of inferences, $\{g_1(Y), \ldots, g_n(Y)\}$, and the app provides its utility requirements as a *whitelist* of inferences, $\{f_1(Y), \ldots, f_n(Y)\}$. The privacy mechanisms are designed to ensure the app can effectively compute whitelisted inferences to some degree of accuracy, but where the app cannot draw the blacklisted inferences. Ideally, any data shared with an app should not reveal any more information than what is already known to the app about the blacklisted inferences from prior (population-scale) knowledge or side-channels. We remark that this is a general formulation of the privacy problem, and that the previously mentioned privacy mechanisms such as anonymization and protection against reconstruction attacks can be thought of as carefully chosen blacklist inferences.

2.1 Information Disclosure Regimes

The various possible tradeoff points for the utility and privacy objectives define a spectrum of information disclosure regimes that a user can operate in. At one extreme, corresponding to zero disclosure, the user shares no information at all, ensuring complete privacy but at the cost of complete loss in utility. At the other extreme, corresponding to full disclosure, all information is shared. Now the user achieves utility at the cost of complete loss of privacy. Each point in this spectrum is realizable by using an appropriately designed privacy mechanism, now we discuss some two operation points of particular interest.

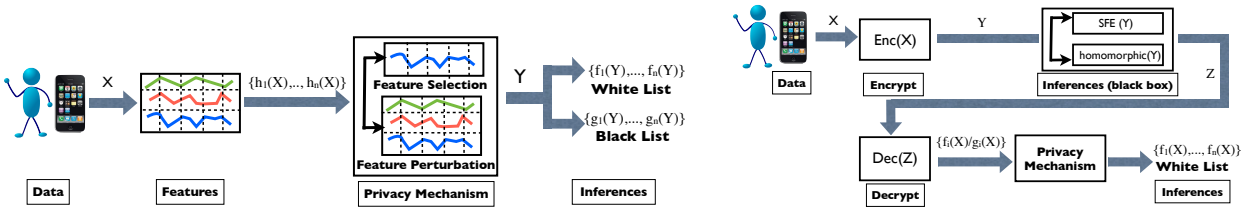
1. Maximum Privacy Disclosure (MaxP): We release information (some transformation of X) such that only the desired utility (whitelisted functions, and consequences inferable from them) can be computed from the released information. This point corresponds to targeted disclosure.

2. Maximum Utility Disclosure (MaxU): We release information which preserve all characteristics of X, except those which can be used to violate privacy (blacklisted functions). This point corresponds to targeted hiding.

2.2 Realization of the Privacy Mechanisms

We define a privacy mechanism as a two-step process: first identifying the data to be shared (e.g., the subset of features, data samples, inferences, data types, etc.) and then applying obfuscation to the data before sharing. Below, we enumerate a set of potential privacy mechanisms.

1. Feature Selection: Instead of the high-dimensional data X, from which information flow is hard to control [19, 18] we extract a set of features $F = \{h_1(X), \ldots, h_n(X)\}$ and use them to represent the data in a lower-dimensional space.



(a) Feature selection and perturbation steps for MaxP and MaxU, respectively.

(b) SFE and Homomorphic encryption for black-box computation of whitelisted functions.

Figure 2: Different realizations of privacy mechanisms.

The functions $h_i(X)$ can represent features like mean, variance, Fourier coefficients, etc., extracted from the data samples over time. Inferences typically operate in the feature space and use a subset of F to perform their classification.

To implement MaxP, we observe that by sharing features we can better control the information shared. The privacy mechanism (see Fig. 2(a)) selects a subset of features required by the whitelisted inferences but which do not contribute to the blacklisted ones. The obfuscation step either suppresses all the other features and shares only the selected features or synthesizes data X' preserving only selected features (and their consequences) and nothing else. This mechanism requires the app to share information regarding the features the inferences depend on.

2. Sharing whitelisted inferences: Another privacy mechanism which also implements MaxP is that of sharing whitelisted inferences (or suppressing blacklisted ones). The idea is to compute the inferences on the phone, obfuscate the results such that they do not reveal any information about the blacklisted inferences and share the obfuscated results instead of X. For this to work, the apps need to provide the exact implementation of the inference algorithm to the user, which may be proprietary and difficult to share. An alternate strategy for evaluating the whitelisted inference functions is to use cryptographic techniques. We suggest two such techniques (see Fig. 2(b)).

- One-sided Secure Function Evaluation (SFE) can applied (using, e.g., Yao's garbled circuit [27]) to evaluating the inference function. Both parties provide their inputs (the user provides her sensor data, and the app the inference function), and the function is evaluated. Since the protocol is one-sided, only the user obtains the result of the computation; and the app knows nothing about the user input. The user can then obfuscate the result before sharing it with the app.

- Homomorphic Encryption [8] allows computation to be carried out on the cipher text directly, yielding an encrypted result of the operations performed on the plain text. The user performs homomorphic encryption on the data and sends it to the app, which can then perform function evaluation on the encrypted data and return the encrypted result to the user, who decrypts it to obtain the result. The second step is to perform obfuscation of the result before sharing with the app.

While the above techniques allow computation of the inference functions without their disclosure, there is no way for the user to know if the results computed are for the whitelisted inferences only. Thus the privacy mechanism must use other techniques (such as zero knowledge proofs [10], random spot checks, etc.), to ensure that the correct functions are being evaluated. In addition, while feasible in theory, these techniques are extremely computationally expensive and thus energy-intensive.

3. Random Projection: Following this mechanism (see Fig. 3), we share projections of the features instead of the features themselves [16]. That is, we project the features into a lower dimensional space before sharing. To ensure that privacy is maintained, the transformation is kept private and is known only to the user.

For utility goals, the user furnishes training labels so that the app can learn a classifier, based on the projected features and associated labels, for the whitelisted inferences (and their consequences) but nothing else. In order to learn the labels in the embedded space, the key property required is that pairwise distances between points in the original feature space be preserved. Fortunately, when the transformation is derived from randomly generated basis vectors drawn from an i.i.d. normal distribution, the Johnson Lindenstrauss lemma states that this property holds with high probability when the dimensionality of the new projected feature space satisfies a certain size constraint [13, 16].

This mechanism eliminates the need to know a priori the mapping between the inferences and features as required by the feature selection approach. It places a significant burden on the app, however, which must now learn the classifier or the whitelisted inference. An advantage of using this mechanism is that we can guarantee privacy when there is no side-channel information, as only the whitelisted inference labels are shared.

4. Feature Perturbation: We use this mechanism to realize MaxU (see Fig. 2(a)). We select and transform a specific set of features, and share everything else. For example, for an audio signal we can choose pitch as the feature to transform and use perturbation to obfuscate it. While inferences such as identification of the speaker, which rely on pitch, are affected, other inferences not depending on pitch remain accurately computable. One of the drawbacks of this mechanism is that it does not protect against blacklisted inference functions, which can learn a classifier using the set of released features instead of the transformed ones.

3. DATA FLOW SCENARIOS

A privacy framework implementing (a possible subset of) the above privacy mechanisms must fit into existing state-of-the-art mobile platforms. We investigate the different placement points of such a framework by taking into account the various data flow scenarios between a user and the apps.

We use the Android OS [1] (see Fig. 4(a)) as representative of a state-of-the-art mobile system. To simplify our

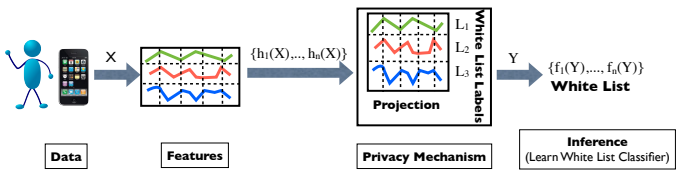


Figure 3: Random Projection for sharing whitelisted functions. We share both projection and the corresponding whitelist labels.

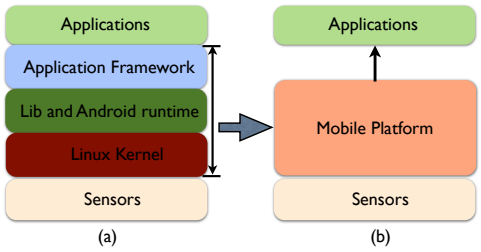


Figure 4: Abstract model of a phone.

presentation, we derive an abstract model that combines the functionalities of the middle layers into a single block and call it the mobile platform. Apps running on the phone form part of the topmost layer (see Fig. 4(b)). We assume that the platform is *trusted* to not leak user information whereas the app layer (running third-party apps) is *untrusted*. Figs. 5 (a), (b) and (c) illustrate the possible data flows. Computation local to the phone is performed within the dotted box. The implementation of the privacy mechanisms will vary depending on the placement of the privacy framework.

If a locally running app (shown in Fig. 5(a)) is completely insulated and does not communicate outside the dotted box then the user need not obfuscate data before sharing. However, static analysis of app code and information flow tracking techniques [7] have revealed the existence of side channels through which apps leak information to the cloud [9]. To prevent such attacks the framework needs to be placed between the mobile platform and the app and would require changes to the platform code. For a cloud-based app (Fig. 5(b)), the framework can be included as part of the client implementation. While this would eliminate the need to make platform changes as described for the previous case, it would involve modification of the app clients. In addition, the modified client code would need to be trusted and not leak information. Finally, with a broker-cloud-hosted app (Fig. 5(c)), the privacy framework can be pushed to the trusted server implementing the broker service.

Each placement choice corresponds to a different tradeoff in terms of implementation complexity. Choices in Figs. 5(b) and 5(c) are specific to an app client or a broker, and hence might require significant duplication of implementation effort. Also, for Figs. 5(a) and 5(b), the implementation has to be lightweight owing to resource constrained mobile platforms, whereas there is no such restriction when the framework is running on a trusted server as in Fig. 5(c). Without assuming a trusted broker, we implement our framework as part of the mobile platform. We can thus intercept and obfuscate data in all the possible scenarios.

4. ipShield: THE PRIVACY FRAMEWORK

We conceptualize `ipShield`, the inference privacy framework (see Fig. 6) and describe our implementation on an

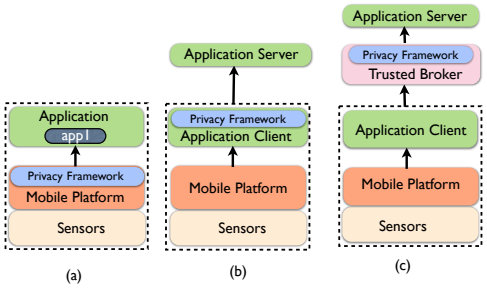


Figure 5: Data-flow paths from user to apps. Data is shared with (a) locally running apps, (b) directly with cloud-hosted apps. and (c) via trusted broker with cloud-hosted apps.

Android-based mobile platform. Prior work on privacy frameworks rely on static privacy policies, or use information flow techniques to detect potential leakage from apps and apply binary policies of complete access or no access to data at all [1, 2, 7]. In comparison, `ipShield` makes two main contributions. First, it implements context-aware privacy policies (blacklist and whitelist specification). Broadly, context refers to a combination of the current physical (e.g. walking, running, still, smoking), location (e.g. indoor, outdoor, office, home), social (with friends, in meeting), and even psychological (e.g. stress) state of a user and can be inferred from a variety of sensor measurements. There is active research towards creating an operating system service [5, 26], which would provide apps with *contexts* rather than sensor data. Context-awareness allow users to define dynamic fine-grained privacy policies depending on current context – an improvement over the current binary and static policies. Since the user wants to protect against blacklisted inferences, possibly when in certain context states, other contexts are ipso facto safe for data release. Second, `ipShield`, uses a graphical model to capture initial adversarial knowledge and its subsequent increase with each disclosure. It then uses the model to determine the level of obfuscation required before releasing the data. We present a case for model-based privacy and follow it up with the design and initial implementation of the privacy framework.

4.1 A Case for Model-Based Privacy

The degree of obfuscation required depends on the adversary's capabilities. Prior work has shown that human behavior, and thus, we can assume, behavioral inferences, exhibit significant correlation [14], which can be captured using graphical models [20, 12]. We assume a model-based adversary that maintains a belief on the blacklisted inferences based on prior knowledge and continuously updates the model parameters by observing the shared data.

4.1.1 Adversary Model

The maximum amount of information that can be extracted by an adversary is quantified by the mutual information between the whitelisted and the blacklisted inferences. To compute the mutual information, we need to learn the joint distribution of the two, which can be done using sophisticated graphical models. The mutual information gives an upper bound on the amount of information which can be possibly extracted by the adversary.

We assume that the adversarial power is captured using graphical models such as a Dynamic Bayesian Network (DBN) or a Markov Chain. The states of the model corre-

spond to the different inferences that could be made using the shared data. Prior knowledge of the adversary from auxiliary sources is expressed as the prior probability on the occurrence of a state and the transition probabilities on the edges. We track the change in adversarial knowledge by updating the probabilities with every data release.

4.1.2 Learning the Obfuscation Function

The obfuscation functions are used to transform the data so that (a) the adversary cannot make the blacklisted inferences; (b) the data utility is preserved; and (c) the obfuscated data is *plausible*, i.e. not so obfuscated that the data points become outliers which can be easily filtered by the adversary. However, the obfuscation function only provides privacy against the graphical model used for the adversary. This requires that the model be powerful and capture data dependencies effectively.

Depending on the adversary's belief we define three different types of obfuscation actions: (a) Suppression, where data is not released. However, we need to ensure that suppression itself does not increase adversarial knowledge about sensitive state; (b) Perturbation, where structured noise is added to increase uncertainty in the blacklisted inferences; and finally (c) Synthesis, where synthetic data unrelated to the actual data is generated by sampling the graphical model, to ensure plausibility of the obfuscated data.

4.2 Design of `ipShield`

The different layers of `ipShield` are shown in Fig. 6 and are explained below.

Sensors: This layer provides access to built-in sensors on the phone (e.g., accelerometers, GPS, microphones, camera), or external body-worn sensors such as a galvanic skin response sensor, EKG sensors, or other virtual sensors such as the calendar (providing event schedules), battery (providing power status), feature sensor extracting various features such as mean, variance, etc. from actual sensor measurements (e.g. probes in Funf [3]).

Context Framework: This layer collects data from the sensors and uses it to identify the current *context state* of the user. We have implemented this layer as an extension to the open-source sensor data collection and dissemination library Funf [3]. For example, an activity context can be a decision tree classifier that was trained offline. At runtime, accelerometer data is given to the classifier, which extracts features such as mean, variance, and Fourier coefficients, and uses them to classify the data into activity states such as *walking*, *running*, and *still*.

Inference Framework: The inference framework takes the current state, which recall is a set of contexts, and uses them to compute inferences. This is again done using classifiers, which are trained offline using contexts and user-provided labels as training data. For example, presence at the location of a religious place, together with time of day and day of week, could be used to make an inference about religious preferences. These inferences are part of the black list and white list specified by the user and the apps, respectively.

Privacy Firewall: The privacy firewall comprises of three different subsystems. First, the graphical model, which is as mentioned in Section 4.1. Second, the user preferences subsystem, which allows users and apps to specify the black list and white list of inferences, respectively. The third subsystem contains the rules and the obfuscation blocks. The rule block contains the set of privacy policies (similar to rules using iptables for configuring network firewalls), which specify the obfuscation action on the sensor data when a specific context state is true. These policies can be either configured by the user or derived from the white and black lists using the graphical model. There are two ways a user can configure these lists. First, we envision that similar to network firewall config files, or spam filtering, a user can obtain such config lists published by privacy experts and personalize them according to her preferences. Second option that is currently being researched is the use of crowdsourcing to understand user expectation of privacy and utility of popular apps used on the phone [15].

The obfuscation block implements the different actions. If the obfuscated data does not increase the adversarial knowledge about the sensitive inference, data is released, else, it is subjected to further iterations of obfuscation before it meets the privacy and utility requirements.

Application: The privacy firewall is the point at which data is shared with the apps. While most of the current apps require sensor data directly, there has been a steady growth in context-aware apps which take contexts as input instead of sensor data. Thus, the firewall should implement interfaces for sharing both obfuscated sensor data as well as derived contexts.

4.3 Prototype Implementation and Use Case

In [21] we provided an implementation of the changes on the Android platform required to enforce the context-aware obfuscated sensor data sharing. For model-based privacy, we created a prototype of a DBN on the Android platform [4] and used it to determine when to suppress or release data. We are currently working towards a privacy rule specification framework for user preferences.

As use case, we consider an example of selective suppression of features extracted from accelerometer data. *Activity level* of a user, is a binary decision of being active or inactive, and can be inferred via decision trees directly from features over windows of accelerometer data. The adversary model is also a decision tree that infers the *activity type*, such as walking, running, biking, from the same windows of accelerometer data (E1 in Section 1). Thus, the white list here is to perform activity level detection and the black list is to prevent the detection of specific activity type. The obfuscation we perform is selective suppression of features (we release entropy of Fourier coefficients) such that the *activity type* inference is no longer inferable via a decision tree based adversary.

5. LIMITATIONS AND FUTURE WORK

The general privacy problem of precluding a set of behavioral inferences from being made while ensuring that others *can* be made involves a complex interaction of information theory and machine learning. The problem is well defined only when the whitelisted inferences do not completely overlap the blacklisted inferences, in which case releasing even the whitelist inferences themselves would violate privacy. For well defined settings, the challenge lies in finding the right feature subset or data transformation which will have an acceptably low mutual information with the blacklisted inferences.

Our framework offers a context-aware model-based solution to the problem. The ability of the model to account for the relationship between a variety of inferences, learning

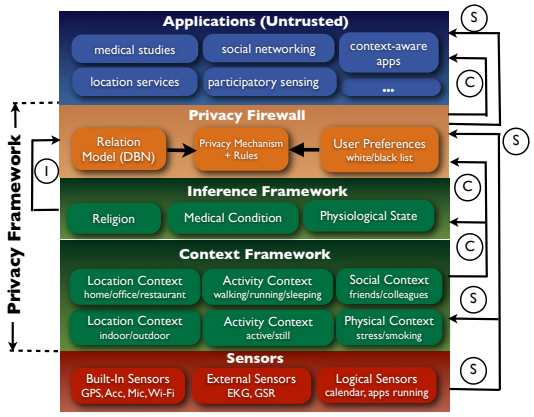


Figure 6: Inference Privacy Framework. S = sensor data, C = context state, I = inference.

accuracy based on training data, and finally the feature set corresponding to the whitelist inferences are all key to the success of the framework. In addition, apps which use data that manifest dependencies over long periods of time, maybe hard to protect against, owing to order constraints on the graphical model.

Finally, our adversary model is limited because it does not account for the relationship between the app producers. A possible enrichment to the model could be to augment it with information from social networks which model personal relationships. In the future, we aim to incorporate these into our system for a better privacy experience.

Acknowledgement

Research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defense and was accomplished under Agreement Number W911NF-06-3-0001. This material is also based upon work supported by the NSF under awards CNS-0910706 and CNS-1213140. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF or represent the official policies of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defense or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon. The first and second authors are supported by the Qualcomm Innovation Fellowship award for 2011-2012. We would also like to thank our shepherd James Scott for his valuable suggestions and guidance.

6. REFERENCES

[1] `http://http://developer.android.com/guide/basics/what-is-android.html`.
[2] `github.com/gsbabil/PDroid-AOSP-JellyBean`.
[3] `http://funf.org`.
[4] S. Chakraborty, K. R. Raghavan, and M. Srivastava. Poster: Model-based context privacy for personal data streams. CCS, 2012.
[5] D. Chu, A. Kansal, J. Liu, and F. Zhao. Mobile apps: it's time to move up to condos. HotOS, 2011.
[6] C. Dwork. Differential privacy: a survey of results. TAMC, 2008.
[7] W. Enck and et. al. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. OSDI, 2010.
[8] C. Gentry and S. Halevi. Implementing gentry's fully-homomorphic encryption scheme. EUROCRYPT, 2011.
[9] C. Gibler, J. Crussell, J. Erickson, and H. Chen. Androidleaks: automatically detecting potential privacy leaks in android applications on a large scale. TRUST, 2012.
[10] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. STOC, 1985.
[11] P. Golle and K. Partridge. On the anonymity of home/work location pairs. Pervasive, 2009.
[12] M. Götz, S. Nath, and J. Gehrke. Maskit: privately releasing user context streams for personalized mobile applications. SIGMOD, 2012.
[13] K. Kenthapadi, A. Korolova, I. Mironov, and N. Mishra. Privacy via the johnson-lindenstrauss transform. *CoRR*, abs/1204.2606, 2012.
[14] E. Kim, S. Helal, and D. Cook. Human activity recognition and pattern discovery. *IEEE Pervasive Computing*, 2010.
[15] J. Lin, N. Sadeh, S. Amini, J. Lindqvist, J. I. Hong, and J. Zhang. Expectation and purpose: understanding users' mental models of mobile app privacy through crowdsourcing. UbiComp, 2012.
[16] K. Liu, H. Kargupta, and J. Ryan. Random projection-based multiplicative data perturbation for privacy preserving distributed data mining. *IEEE Trans. on Knowl. & Data Eng.*, 2006.
[17] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. L-diversity: Privacy beyond k-anonymity. *ACM Trans. Knowl. Discov. Data*, 2007.
[18] A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse datasets. In *IEEE Symposium on Security and Privacy*, 2008.
[19] A. Narayanan and V. Shmatikov. Myths and fallacies of "personally identifiable information". *Commun. ACM*, 2010.
[20] H.-S. Park and S.-B. Cho. Predicting user activities in the sequence of mobile context for ambient intelligence environment using dynamic bayesian network. In *ICAART*, 2010.
[21] K. R. Raghavan, S. Chakraborty, and M. Srivastava. Override: A mobile privacy framework for context-driven perturbation and synthesis of sensor data streams. PhoneSense, 2012.
[22] L. Sankar, S. Rajagopalan, and V. Poor. A theory of utility and privacy of data sources. ISIT, 2010.
[23] M. Srivatsa and M. Hicks. Deanonymizing mobility traces: Using social network as a side-channel. CCS, 2012.
[24] L. Sweeney. k-anonymity: a model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 2002.
[25] T. Vila, R. Greenstadt, and D. Molnar. Why we can't be bothered to read privacy policies models of privacy economics as a lemons market. ICEC, 2003.
[26] T. Yan, D. Chu, D. Ganesan, A. Kansal, and J. Liu. Fast app launching for mobile devices using predictive user context. MobiSys, 2012.
[27] A. C.-C. Yao. How to generate and exchange secrets. SFCS, 1986.

Cloud Displays for Mobile Users in a Display Cloud

Lars Tiede, John Markus Bjørndalen, and Otto J. Anshus
Department of Computer Science
University of Tromsø, Norway
lars.tiede@uit.no, jmb@cs.uit.no, otto@cs.uit.no

ABSTRACT

The display cloud model allows users to select local and remote programmable displays, and add them to a user specific cloud display where the user can arrange them freely. On a cloud display, the abstraction representing remote graphical content is termed a visual. It can be positioned and resized freely. Wherever a visual touches a part of the cloud display with physical displays present, the physical displays will show the corresponding graphical content of the visual. The physical displays can simultaneously show several visuals from one or many users.

The display cloud approach is suitable for public environments because we do not allow user customization of the displays, a user does not have to expose any data except the actual graphical content to the display computers, and he does not have to go through the displays to do user interaction with his resources. Mobile devices have an essential role in achieving this. They provide, for each user, the means to detect displays, to add displays to the user's cloud display, to manage displays and visuals in a cloud display, and to interact with visuals.

An insight is that the display cloud model is maximally decentralized between users, and maximally centralized per user. We conducted a set of experiments on a prototype using 28 display computers with up to 21 users. The results show that the prototype reacts interactively fast for each, and scales well to many users.

Categories and Subject Descriptors

H.5.2 [**Information interfaces and presentation (e.g., HCI)**]: Miscellaneous

Keywords

ubiquitous displays, display clouds, cloud displays

1. INTRODUCTION

The research problem we focus on is how to let a user display content produced on his own computers onto one or several displays both local and remote to the user. The research challenges are to understand how to do this (i) in a scalable way with regards to performance metrics like frame rates, interactive latency, and consumed bandwidth for both a single and multiple users when the number of displays increase, (ii) in a simple way so that a user can rapidly display what he wishes, (iii) in a flexible way with regards to tiling together several displays to create a larger and higher resolution display, (iv) in a secure way demanding no or very little more trust from the user than what he has already given elsewhere, using his own computers.

The methodology applied is systems research where we research possible architectures, designs and implementations of prototype systems, and document the performance characteristics of at least one such prototype. We propose, and have partially implemented, the Display Cloud approach. A user with a mobile device can easily configure a *cloud display* composed of one or many programmable displays from a loose set of displays called a *display cloud*. Using the mobile device, the user can then securely and scalably display content produced at or controlled by his computers onto the cloud display. A user can flexibly define many content entities, termed *visuals*. When a user moves from one display to another across a room, building, city, country or continent, the visuals can follow the user or be displayed wherever the user wants as long as the displays are a part of the user's cloud display.

As a case, let's assume that Amy meets her friends at a coffee shop and wants to show them pictures she has on her home computer. There is a large display cloud enabled display above the table. Amy takes her smartphone, starts the display cloud app, and scans a unique ID, a QR code, on the display. The app then connects to the display and Amy sees it represented as a rectangle on her phone. The new display has become a part of her phone's cloud display and can be dragged and positioned relative to any other display in her cloud display using the phone. Amy then uses the phone to select a picture viewing application on her home computer as the source for a visual. The visual is represented on the phone as a rectangle, and Amy drags the rectangle onto the new display's rectangle to make it visible. The display cloud system starts a viewer on the new display, and it begins to show the content produced by Amy's home computer. Through the smartphone, she can freely reposition and resize the visual content on the large display, and interact with the home computer application. Amy's friends can now watch the pictures on the large display. After Amy is done, she removes the coffee shop display from her cloud display, and all content delivered from Amy's home computer disappears. If Amy forgets to manually delete the display from the cloud display, it will automatically happen when the smartphone discovers that Amy has moved away.

We assume that a user will always have a mobile device available. We use the mobile device to (i) determine the location of a user, (ii) detect physically nearby displays, (iii) quickly set up one

HotMobile'13, February 26–27, 2013, Jekyll Island, Georgia, USA.

or several displays for temporary use, (iv) establish connection between the mobile device, the displays and a user's remote PC so that user input is transferred to the PC without involving the display(s) and display output is transferred onto the displays directly from the PC.

We observe that private and public spaces have an increasing number of displays. The expected proliferation of Android- and iOS-based consumer televisions and displays make it realistic to expect cheap programmable displays to be ubiquitous in many environments.

The trend towards open programmable displays everywhere combined with small always present mobile devices provides for the technologies needed to let us do better than today, whether we are at home, at work, or travelling: information may be displayed everywhere, in sizes suitable for multiple viewers and a lot of information may be displayed simultaneously.

The way we perceive ubiquitous displays is different from the traditional way of perceiving displays. Traditionally, displays are output devices connected to a single computer and represent a single, closed area on which the computer places all its visual content. This one-to-one relationship between computers and displays makes it harder to use the displays in the settings we described above. We need to easily compose several physically close displays into one larger display to get higher resolution and larger size. We also need to easily move display content between displays when we move from one place to another.

There are existing approaches to use more than one display at a time, either by connecting several displays to one computer in a multi-monitor setup, or by combining several displays and computers into a configuration such as a tiled display wall[9]. With these approaches, multiple displays become accessible, and display content can be split between several displays. However, these approaches typically assume a fixed number of displays in one room, and will not support a user moving outside of the room to other displays. Limited scalability with regards to the number of viewers and the number of displays when using VNC[16]-style of centralized control of the displays is documented in [19]. While VNC can support many users viewing the same low resolution desktop, the frame rate will rapidly drop as the resolution of the desktop increases.

We can avoid the limitations and drawbacks described above and achieve better functionality and scaling if we perceive a set of displays as a continuum; an open, distributed, and decentralized display surface. We term this a cloud display. Users can define their own cloud display by composing local and remote displays from the display cloud - the set of displays with the necessary functionality to be enrolled into a cloud display. A cloud display is flexible with respect to the number of physical displays that are part of it, the number of displays that are actively displaying content, and the spatial arrangement. On a cloud display, a user can put any kind of graphical content produced at his local and remote compute resources, termed visuals. Visuals that are currently supported by our prototype are VNC desktops and images fetched from web servers. Other technologies we expect to support include video streaming and desktop sharing approaches such as Apple's AirPlay.

To achieve protection, security, and ease of use for a user when he moves between displays, we use his mobile device to incorporate nearby physical displays into the user's cloud display. A key point is that displays only have access to the graphical output, and then only through temporary capabilities so that when a user session terminates, the display(s) cannot continue to pull in data from the user's PC. Using the mobile device for user input means that we do not have to trust the displays; a display cannot easily snoop on the user's input and capture passwords or other sensitive information entered into applications.

We distinguish the Display Cloud model from research on public displays like those in [2], [3], and [14] in that we have a machine centric focus. We have not investigated how users react to different ways of doing and using public apps. Furthermore, we propose or assume no special public display apps, and we have no floor control system. We have only researched and documented (i) a system making one or several displays into shared displays that users can freely compose and use as display surfaces. It is entirely up to the users what they display and where on the displays they display it. (ii) A system where mobile devices play a crucial part to achieve security by letting the users only trust what they already trust: their own computers. (iii) A system where mobile devices provide user input to the computers running the application(s) that produce output for the displays.

We believe that the Display Cloud system can be used on public displays for advertisements and informational purposes as well as by individuals briefly needing larger displays.

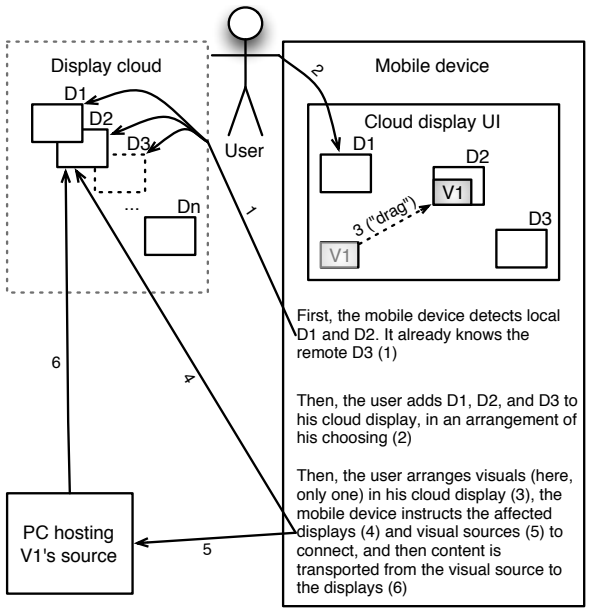


Figure 1: Display cloud, cloud displays, and visuals

2. USAGE SCENARIOS

Figure 1 illustrates the concepts used in the following usage scenarios. The scenarios hint at some functionalities and features not yet available in the prototype. However, we have found the scenarios useful in describing the range of possibilities, and they help us identify issues to be solved.

Scenario 1: In the Lab, Giving a Presentation. Ken, a visiting researcher, and his hosts meet in a lab for a presentation. The lab has a tiled display wall comprised of many displays. Ken uses his smartphone to detect and add the displays of the display wall to his cloud display, and arranges them into rows and columns in the same way as the display wall. Ken has already configured a visual using his cloud-hosted virtual machine as the visual source. On the smartphone, he now selects where on the cloud display the output should take place by moving the visual over the display wall's displays. His smartphone instructs the displays overlapping with the visual to start a viewer and display the remote content. Ken can

interactively resize and reposition the output to cover less or more area of the display wall, and the smartphone will direct the displays accordingly. After the presentation is finished, discussion starts and others move some of their own visuals to the display wall as well. Ken can resize his visual to make more display space available for the others' visuals. When the discussion is over and the participants leave the lab, their smartphones detect that they are no longer in the lab, and each display cloud app automatically detaches the displays from its respective cloud display, and their computers are told to serve no more content to the display wall.

Scenario 2: At the Hotel, Doing Remote Lecturing. Ken missed his flight back, and now has to give a lecture from the hotel room. He adds the large displays in the lecture hall as well as the hotel room display to his smartphone's cloud display. He defines three visuals: one with his laptop presentation as visual source, one with his laptop's camera as source, and one with the lecture hall's audience-facing camera as source. In the cloud display user interface, he moves the first two visuals to the lecture hall displays, and the last one to his hotel room's display. The students can now watch the lecture slides as well as a video of Ken on the auditorium's displays, and Ken can watch the students on the hotel room's display.

Scenario 3: At the Mall, Hanging Out. A group of teenagers meets at a mall. The mall has large displays everywhere. It also has sensors tracking customers. Both are made available to the mall shops and are paid for through subscriptions. For a customer, a display is complimentary to use for a brief period when standing next to it. The displays are used by the shops to display advertisements and coupon visuals, and by customers to display both their own visuals and interact with advertisement visuals.

When the mall's sensors and related analytics detect the group of teenagers, advertisement visuals looking for groups of teenage customers swarm towards displays near the teenagers and follow them around the mall. A teenager interested in an advertisement on a nearby display uses a smartphone to rapidly include the display to the smartphone's cloud display. Advertisement visuals, but not other customers' visuals, on that display automatically become available for selection. From the cloud display user interface, the teenager selects the interesting advertisement visual, and an instance of the advertisement visual is created for the teenager. The teenager can now interact with this instance of the advertisement visual. Other teenagers get their own instance, allowing each to browse and purchase products independently from each other.

After a while, the teenagers move on, their smartphones detect this and instruct the remote computers to stop sending data to the displays left behind. Alternatively, the smartphones can automatically add and delete nearby displays to their cloud displays, allowing the users private and advertisement instances of visuals to follow them around, moving from display to display around the mall. In both cases the visuals live on, and can be displayed again on other displays.

3. DISPLAY CLOUD ARCHITECTURE

To make a set of displays into a display cloud, and to make cloud displays from the display cloud, there are several primary functionalities we have discovered that we either need or should not have.

A display must be able to interact with a mobile device. The minimum functionality that must be in place for a display is to let a mobile device customize it, either by setting parameters for functionality already present, or by accepting new functionality given to it by the mobile device.

In the first case, the mobile device must trust that the display does not misbehave by, say, copying graphical content it sees to a third party. However, it cannot touch the user's original remote data or discover user passwords because these are handled exclusively between the mobile device and the user's remote computers. In the second case, the display must trust the mobile device as well. The mobile device can make it misbehave in obvious ways, like displaying unintended information from the internet, or adding the display computer to a botnet. Sandboxing can help reduce such dangers. Perhaps one day we will understand how to let the display figure out what an uploaded functionality will actually do, and based on this reject it or not.

Presently, we use the first approach, preinstalled functionality, recognizing that the mobile device cannot trust the display. This is a simple approach with easy to understand implications: the visuals can be compromised, but nothing else.

Functionality making all user data received by displays disappear from the display when the user wants to or when the user moves away from the display. The display will delete all user data it has received when the mobile device tells it to, and when the user moves beyond a certain distance from the display. Further, when a user quits using a display, the display must not be able to continue pulling in data from the user's remote computer. This is solved by letting the remote computer refuse further requests from the display, either when instructed by the mobile device or if the computer loses communication with the mobile device.

Mobile device functionality enabling it to dynamically discover nearby displays. This can be done in several ways using technologies such as visual tags, NFC, or Bluetooth. We are currently looking into using QR codes on or next to the displays to let mobile phones discover them and retrieve the URL or transport level network address of the displays.

Mobile device functionality to compose displays into a cloud display. The user tells the mobile device how the displays should be arranged. The mobile device tells each display what it should display. This is done at suitable frequencies, say, 25 times a second. Interestingly, the displays have no knowledge of each other and do not interact.

Further required mobile device functionality is to instruct the remote PC about making visuals available only to the relevant displays in the cloud display, to define clones of visuals so that the same content can be displayed on several displays, and to enable the user to interact with the remote computer through the mobile device.

A user's remote computers have functionality to provide visuals to the displays in a cloud display. Several approaches are possible: (i) Each display pulls in from the remote computer what the mobile device tells it to display. (ii) The remote computer pushes to a display what the mobile device tells it to push. (iii) The mobile device itself either pulls in visual content from the remote PC, or it tells the PC to push the content to it. The mobile device then acts as a proxy for the displays in either a push or a pull mode.

Each approach has advantages and disadvantages which we don't have the space to expand on here. Presently, we use the first approach.

4. PROTOTYPE

We have developed a functioning prototype of a display cloud. Many core functionalities are already implemented, including creating a cloud display, displaying visuals on the physical displays of the cloud display, and moving them smoothly on and between displays.

The prototype currently supports two approaches to transporting graphical content from a remote computer to the displays comprising a cloud display. In the VNC[16] approach, each display starts a

customized VNC viewer which then requests content from a VNC server running on a remote computer. We also use an approach where each display starts a simple picture viewer that requests images from a remote computer via HTTP.

Displays run a *display daemon* to make their functionality available to mobile devices. It listens for network connections from mobile devices, and starts visual viewers on behalf of them. Visual viewers only run as long as a user uses them; they are started and terminated on demand. This introduces an overhead when viewers are started, but it makes sure that no user state is left behind when the user stops using a display. It also minimizes persistent resource usage on the displays. Presently, the prototype does not discover when a user moves away, and the set of available displays is statically configured.

The *user controller* composes the cloud display, i.e., it deals with arranging displays, and creating, cloning, and placement of visuals. The user interacts with it through a user interface. At the moment, the user controller and the user interface run on a PC because we haven't yet ported them to mobile devices, but the infrastructure has the necessary support for mobile versions. Most user interaction is currently scripted to provide repeatability for experiments.

The *visual controller* manages a visual, i.e., it instructs displays to start and stop viewers, and it instructs visual viewers which region of the visual must be shown and where to display it. There is one visual controller instance for each visual that is managed by the user controller. The visual controller will be responsible for authorization features and proper interaction between users and visual sources when we add mobile devices.

The prototype is implemented in Python, except for the viewers which are implemented in C. The VNC viewer is a modified TightVNC viewer, and the image viewer is written from scratch. All components currently run on Linux.

The visual sources are unmodified TightVNC servers and HTTP servers. VNC servers support multiple viewers out of the box, so we only needed to instruct the viewers to request their separate regions to support splitting of a desktop to multiple physical displays. Our VNC clients can scale pixels to support different resolution displays. Other remote desktop and content streaming systems that we currently consider adding to our prototype will use different techniques for multi-resolution support.

5. EVALUATION

We report on a subset of the experiments we have conducted and their results. When changing a visual's position 30 times a second, we measured (i) how much time it took to move it on a single display and between displays, (ii) the consumed network bandwidth, and (iii) the CPU load on the display computers. We varied the number of users from 1 to 21. Each user had a single visual.

The computers were connected through a 1GBit switched Ethernet. To emulate an environment with many displays, we used 28 quad-core PCs with Linux, modified TightVNC viewers and 28 displays. To easily view and control the experiments, we used PCs that were located in a single room. We used the displays as if they were arranged into one row. To emulate a user's remote PC, we used a single core PC with Linux and TightVNC server 1.3.10. When increasing the number of users from 1 to 21, we used a cluster of identical PCs so that each user had their own remote PC. To emulate a mobile device, we ran a Python process on a display computer. User input was scripted to ensure repeatability: a mobile device process will 30 times a second tell a display to move a visual. The number of mobile devices was increased from one to 21, spreading them out so that a display computer would not host more than one simulated mobile device.

The results show that moving a visual takes about 8ms on a single display, and typically 150ms when a visual moves relatively slowly (below 3 m/s on our displays) from one display into another. The longer time for cross-display movement of a visual is because the mobile device tells a display to boot an instance of the visual viewer every time a visual enters a new display. When a visual moves faster than 3m/s, the time it takes to cross between displays is much longer, in some cases taking several seconds. When this happened, we observed that VNC was the bottleneck, spending most of the time doing protocol initiation. We suspect this is because we do too frequent connection establishments and teardowns.

When increasing the number of users, each with one visual, from 1 to 21 in the display cloud of 28 displays, the time to move each visual increased insignificantly.

A single visual consumed 1MB/s bandwidth, adding to 21MB/s with 21 visuals being displayed and moved. This is well below the capacity of the 1Gbit/s Ethernet we used. If physical mobile devices had been used instead of emulating them using processes, this would not have impacted the measurements significantly because the data to be visualized does not go through a mobile device, but directly from a remote PC to the displays.

Each mobile device process consumed less than 10% CPU on the display computer where it was running. Based on benchmarks we did, we estimate this to have been about 40% CPU load on a Samsung Galaxy S3, which has a quad-core ARM processor and 1GB RAM, running Android 4.0.4. Using a native application instead of a Python application is likely to be more efficient than the estimated 40%.

We have not reported on frames per second (FPS). FPS is limited by the remote graphics technology we use for the experiments, VNC. VNC frame rates have been reported elsewhere [7, 10]. Frame rates depend upon the number of pixels, the CPU of the VNC server, and available networks. In our setting we see typical VNC frame rates from 1-15 FPS while the viewers were moved and re-drawn at 30 FPS.

6. RELATED WORK

We use the term "ubiquitous display" as has been described by Molyneaux and Kortuem[13], who give an overview over possible technologies for ubiquitous displays, and research challenges.

Work on distributed displays and in particular display walls brought forth approaches to provide big centrally managed virtual displays to programmers and users, for example SAGE[8] and Distributed Multiheaded X[12]. Seamless screen sharing over several displays, taking not only different display resolutions but also geometrical distortions into account, has been investigated by Sakurai et al[17]. These systems usually have a static set of displays, whereas our approach aims at scalability of display usage and sharing to many users in many rooms, using a subset of many, possibly even remote displays.

Beyond dealing with distributed displays alone, many works have focused on enabling collaboration between users on large single or distributed displays through screen sharing and more, for example Dynamo[4], WeSpace[6], Impromptu[1], and Virtually Shared Displays[20]. These particular works differ from ours mainly in that they focus on enabling collaboration including, for some systems, file sharing, between users in one room.

All approaches above do not concern themselves with users composing their own view on the set of available displays, i.e., there is no conceptual equivalent to a cloud display. Mobile devices with their special sensing functionalities do not exist or do not play *essential* roles in their architectures. Several systems use the users' laptops as interaction device and content provider, whereas we sep-

arate these roles (although visuals can be hosted on smartphones, too). This makes our system more useful in a mobile environment because a user only needs to carry a smartphone to pull in content from anywhere. Another difference is that most of the above systems (except Dynamo) are tailored towards private environments or the workplace, not public spaces where devices and users can not be trusted. Furthermore, the referenced papers do not report on scaling with respect to many concurrent users or many displays.

A different approach to making displays available to users more dynamically is Dynamic Composable Computing[22], where logical computers are composed ad-hoc from a set of available devices. Unlike the display cloud approach, DCC goes beyond concerning itself only with display mechanics: to facilitate user collaboration, it also supports interconnecting other services such as different users' file systems and clipboards. When it comes to display mechanics, in DCC, a (stationary or mobile) device's framebuffer can be "connected" to a nearby display. However, DCC has no equivalent to a "cloud display", where users compose a virtual display landscape out of several local and possibly remote displays: while DCC includes the ability to connect several adjacent displays to form one logical display[11], this larger virtual display is managed as one large rectangular framebuffer and can therefore not be arranged as flexibly as displays in a cloud display. Further, displays in DCC are always in exclusive use by one user, whereas in a display cloud displays are always shared, and concurrent use, i.e. different users using different areas on one display, is possible.

A related idea to the former approach is to use virtualization to compose a "virtual platform" out of nearby resources with the STRATUS[5] system. Such a virtual platform can consist of a display driven by some computer, a CPU that is located on another computer (or even the user's mobile device), and other periphery in the network. The user brings a virtual machine using his mobile device, and STRATUS migrates the virtual machine onto a custom assembled virtual platform. This enables the user to not only show, but also host, graphical content, desktops etc. in the local environment, without having to rely on the user's resource-constrained mobile device, or accessing resources over long distances. However, a STRATUS virtual platform is vulnerable if any of the hosting computers are compromised; in our approach, only the shared graphical content and information about how to reach it is shared with public computers. Furthermore, STRATUS does not support swiftly moving graphical content across displays, sharing one display between several users, or using several displays to compose a cloud display.

No approach mentioned so far has its main focus on enabling users to interact with their own applications on *public* displays, which present challenges of their own. An early work in this area is the "personal server" by Want et al[21]. Here, the user's mobile device, a custom display-less prototype, hosts the user's application, makes its functionality available through a webserver, and a browser running on the public display accesses this webserver to show the application to the user. The personal server device the user carries has only very limited user input facilities, so the user normally uses the computer controlling the display for input. In consequence, the public display must be trusted with user input.

Later, when mobile device technology had progressed, Raghunath et al introduced the "Inverted Browser"[15], in which the user's mobile device pushes content to a modified web browser running on the public display. Here, the user uses his mobile device for interaction too: input on the mobile device is being forwarded to the public display. While this avoids using input devices on the public display directly, it does not alleviate the associated security threat: user input can still be compromised by the public display. Both personal server and inverted browser have in common that the user's mobile device plays the role of the application host, so that applications are confined to the resource-constrained mobile device. In the display cloud approach, however, applications can, but do not have to be hosted on the user's mobile device, allowing for more resource-demanding applications.

When cloud computing for hosting users' applications entered the scene, Satyanarayanan et al introduced Cloudlets[18], where virtual machines hosting the user's applications are synthesized near the physical location of both the user and the (single) public display. For this, the Cloudlets system uses on-site compute hardware, for example some computers in a coffee shop. A virtual machine is assembled from a "base" VM image that is available already (for example a vanilla Linux distribution) and an "overlay" image that the user brings along on his mobile device. After booting the assembled VM, the user interacts with it using his mobile device, and the public display shows the VM's display output. This approach enables the user to leverage the computation power of stationary hardware (applications need not be hosted on his mobile device), while at the same time allowing for applications that demand low latency between input device, application, and display (the application runs physically nearby). In our approach, these latencies are indeed higher, as the compute resources hosting visuals are often not co-located with the user and the public displays he is using. However, as there is no restriction in our approach to where visual sources can be hosted, a display cloud user could use cloudlets to host visual sources he then uses in his cloud display. Trade-offs to using cloudlets include that assembling a virtual machine before and tearing it down after use can take a long time, inducing a significant latency for the user. Further, a cloudlet user must trust the on-site computers with his virtual machine.

7. DISCUSSION AND CONCLUSIONS

A primary assumption of the display cloud model is that a user can only trust his own devices and not the display computers, and vice versa. This distinguishes the display cloud model from ubiquitous computing approaches where the environment detects the user and provides interaction mechanisms for him, and sometimes allows the user to upload code to customize the environment. Consequently, we believe that the display cloud model is well suited for public displays with many mobile users.

To achieve good scaling with the number of users, the display cloud model has no management and control bindings between different users' cloud displays. While visuals from different users can share physical displays and networks, no coordination between visuals from different users is done. Implications of this are that the display cloud model has no obvious limits to growth unless too many users end up using the same networks and the same displays. In the first case this can be solved by increasing the network bandwidth. In the second case we observe that it is highly unlikely that very many users will share the same displays because only a handful of users will fit physically around a display, limiting naturally how many visuals it will be asked to display. Even if very many users could use the same few displays simultaneously, they have no reason to do so because the visuals will conceal each other.

For individual users, the display cloud model relies on a strong centralization handled by a feature rich mobile device controlled and trusted by the user. The mobile device takes care of all interaction between the user and the remote computers, between the user and the displays, and frequently, say, 30 times a second, controls the interaction between remote computers and displays. Interestingly, despite all the responsibilities centralized to the mobile device, it is not a bottleneck. This is because the centralization is per user only.

Acknowledgements

Part of this work has been supported by the Norwegian Research Council, project No. 155550/420 - Display Wall with Compute Cluster, and Tromsø Forskningsstiftelse, project No. A2093 - Display Wall.

The authors would like to thank the Administrative and Technical Staff at the Department of Computer Science for valuable help.

8. REFERENCES

[1] J. T. Biehl, W. T. Baker, B. P. Bailey, D. S. Tan, K. M. Inkpen, and M. Czerwinski. Impromptu: a new interaction framework for supporting collaboration in multiple display environments and its field evaluation for co-located software development. In *Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, CHI '08, pages 939–948, New York, NY, USA, 2008. ACM.

[2] N. Davies, M. Langheinrich, R. Jose, and A. Schmidt. Open display networks: A communications medium for the 21st century. *Computer*, 45(5):58 –64, May 2012.

[3] A. Friday, N. Davies, and C. Efstratiou. Reflections on long-term experiments with public displays. *Computer*, 45(5):34 –41, May 2012.

[4] S. Izadi, H. Brignull, T. Rodden, Y. Rogers, and M. Underwood. Dynamo: a public interactive surface supporting the cooperative sharing and exchange of media. In *Proceedings of the 16th annual ACM symposium on User interface software and technology*, UIST '03, pages 159–168, New York, NY, USA, 2003. ACM.

[5] M. Jang and K. Schwan. Stratus: Assembling virtual platforms from device clouds. In *Proceedings of the 2011 IEEE 4th International Conference on Cloud Computing*, CLOUD '11, pages 476–483, Washington, DC, USA, 2011. IEEE Computer Society.

[6] H. Jiang, D. Wigdor, C. Forlines, and C. Shen. System design for the wespace: Linking personal devices to a table-centered multi-user, multi-surface environment. In *Horizontal Interactive Human Computer Systems, 2008. TABLETOP 2008. 3rd IEEE International Workshop on*, pages 97 –104, oct. 2008.

[7] A. M. Lai and J. Nieh. On the performance of wide-area thin-client computing. *ACM Trans. Comput. Syst.*, 24:175–209, May 2006.

[8] J. Leigh, L. Renambot, A. Johnson, R. Jagodic, H. Hur, E. Hofer, and D. Lee. Scalable Adaptive Graphics middleware for visualization streaming and collaboration in ultra resolution display environments. *Ultrascale Visualization, 2008. UltraVis 2008. Workshop on*, pages 47 – 54, Nov 2008.

[9] K. Li, H. Chen, Y. Chen, D. W. Clark, P. Cook, S. Damianakis, G. Essl, A. Finkelstein, T. Funkhouser, T. Housel, A. Klein, Z. Liu, E. Praun, R. Samanta, B. Shedd, J. P. Singh, G. Tzanetakis, and J. Zheng. Building and Using A Scalable Display Wall System. *IEEE Comput. Graph. Appl.*, 20(4):29–37, 2000.

[10] Y. Liu, J. M. Bjørndalen, and O. J. Anshus. Using multi-threading and server update pushing to improve the performance of vnc for a wall-sized tiled display wall. In *Scalable Information Systems*, volume 18 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 306–321. Springer Berlin Heidelberg, 2009.

[11] K. Lyons, T. Pering, B. Rosario, S. Sud, and R. Want. Multi-display composition: Supporting display sharing for collocated mobile devices. *Human-Computer Interaction – INTERACT 2009*, 5726/2009:758–771, 2009.

[12] K. E. Martin, D. H. Dawes, and R. E. Faith. Distributed Multihead X design. Retrieved October 11, 2012 from: `http://dmx.sourceforge.net/dmx.html`, 2003.

[13] D. Molyneaux and G. Kortuem. Ubiquitous Displays in dynamic environments: Issues and Opportunities. *Proceedings of UbiComp*, Jan 2004.

[14] T. Ojala, V. Kostakos, H. Kukka, T. Heikkinen, T. Linden, M. Jurmu, S. Hosio, F. Kruger, and D. Zanni. Multipurpose interactive public displays in the wild: Three years later. *Computer*, 45(5):42 –49, May 2012.

[15] M. Raghunath, N. Ravi, M.-C. Rosu, and C. Narayanaswami. Inverted browser: a novel approach towards display symbiosis. In *Pervasive Computing and Communications, 2006. PerCom 2006. Fourth Annual IEEE International Conference on*, pages 6 pp. –76, march 2006.

[16] T. Richardson, Q. Stafford-Fraser, K. Wood, and A. Hopper. Virtual network computing. *Internet Computing, IEEE*, 2(1):33 –38, jan/feb 1998.

[17] S. Sakurai, Y. Itoh, Y. Kitamura, M. Nacenta, T. Yamaguchi, S. Subramanian, and F. Kishino. A Middleware for Seamless Use of Multiple Displays. In *Interactive Systems. Design, Specification, and Verification*, volume 5136 of *Lecture Notes in Computer Science*, pages 252–266. Springer Berlin / Heidelberg, 2008.

[18] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *Pervasive Computing, IEEE*, 8(4):14 –23, oct.-dec. 2009.

[19] D. Stødle, J. M. Bjørndalen, and O. J. Anshus. De-Centralizing the VNC Model for Improved Performance on Wall-Sized, High-Resolution Tiled Displays. *Proceedings of Norsk Informatikkonferanse*, pages 53–64, 2007.

[20] G. Wallace and K. Li. Virtually shared displays and user input devices. In *2007 USENIX Annual Technical Conference on Proceedings of the USENIX Annual Technical Conference*, ATC'07, pages 31:1–31:6, Berkeley, CA, USA, 2007. USENIX Association.

[21] R. Want, T. Pering, G. Danneels, M. Kumar, M. Sundar, and J. Light. The personal server: Changing the way we think about ubiquitous computing. pages 194–209, 2002.

[22] R. Want, T. Pering, S. Sud, and B. Rosario. Dynamic composable computing. *HotMobile '08: Proceedings of the 9th workshop on Mobile computing systems and applications*, Feb 2008.

Towards Synchronization of Live Virtual Machines among Mobile Devices

Jeffrey Bickford
AT&T Security Research Center
New York, NY, USA
jbickford@att.com

Ramón Cáceres
AT&T Labs – Research
Florham Park, NJ, USA
ramon@research.att.com

ABSTRACT

The mobile computing experience would improve if users could switch seamlessly from one device to another, with both data and computation state preserved across the switch without apparent delay. This paper proposes *VMsync*, a system for synchronizing the state of live virtual machines (VMs) among mobile devices. VMsync seeks to incrementally transfer changes in an active VM on one device to standby VMs in other devices, so as to maintain a consistent VM image and minimize switching latency. However, constraints of the mobile environment make these goals difficult to achieve and raise many research questions. We present our preliminary design for VMsync and a feasibility study aimed at determining how much data would need to be transferred under different mobile workloads and synchronization policies. For example, through experiments with a Xen VM running Android and playing a YouTube video, we show that sending dirty memory pages transfers 3 times more data than sending only the bytes that actually changed in those pages. Overall, we conclude that VMsync is a feasible approach deserving of further research.

1. INTRODUCTION

People increasingly rely on mobile devices in their everyday lives, often multiple devices such as a smartphone and a tablet. The utility of these devices would improve if users could switch seamlessly from one device to another, in particular if they could continue using applications on the second device exactly where they left off on the first device, with both data and computation state preserved across the switch without apparent delay. For example, a user who starts watching a video on a smartphone may want to continue watching the video on the larger display provided by a tablet.

A limited form of such device switching is currently available through per-application data synchronization. For example, Apple's iCloud service synchronizes changes to calendars, address books, and a few other supported applications. Some other applications provide their own synchronization facilities. However, this approach requires separate and often specific support to be built into each application. A general solution that works for all applications would scale better to the rapidly growing set of mobile applications, which already number in the hundreds of thousands [9].

This paper proposes *VMsync*, a system for synchronizing the state of live virtual machines (VMs) among mobile devices. System-level VMs have been widely proposed to improve the security, manageability, and other aspects of mobile computing [3, 5, 6, 8, 12]. In the context of device switching, VMs encapsulate both data and computation state for a complete operating system and all its applications. Therefore, synchronizing VM state between mobile devices automatically synchronizes all application state.

VMsync seeks to incrementally transfer changes in an active VM on one device to standby VMs on other devices, so as to maintain a consistent VM image and minimize switching latency. This way, when a user switches between devices, there should only be a small amount of data left to transfer before the VMs on both devices become fully consistent, and the switch can be made quickly enough that the user will not notice any delay.

However, constraints of the mobile environment make these goals difficult to achieve and raise many research questions. For example, intermittent connectivity may delay dissemination of changes. Similarly, bandwidth, processing, storage, and energy limitations introduce challenging tradeoffs between simple schemes that transfer complete memory pages or disk blocks, and more sophisticated schemes that transfer only the portions of those pages and blocks that have actually changed.

This paper presents our early efforts towards a complete VMsync design, implementation, and evaluation. After discussing the most relevant prior work, we describe our preliminary design. The VMsync architecture involves a daemon running outside the guest VM on the active device. This daemon inspects the memory and file-system state of the VM and sends recent changes to a server in the cloud. The server then forwards the changes to standby devices.

Finding appropriate policies for how to represent changes and when to send them are central research issues in this work. For example, should the device send whole dirty pages and blocks, or only the changed portions? Should the device send periodic checkpoints or wait for contextual hints? Should the server forward changes as it receives them, or post-process them to reduce the amount of data sent to standby devices?

HotMobile'13, February 26–27, 2013, Jekyll Island, Georgia, USA.

We also present a feasibility study aimed at determining how much data would need to be transferred to maintain a consistent VM image across devices, under different mobile workloads and synchronization policies. We use the size of these data transfers as a rough proxy of various costs incurred during VM synchronization: bandwidth, latency, and energy. We believe that bandwidth, latency, and energy costs will be to some degree proportional to data transfer size. We plan to explicitly measure these different costs in a future full-fledged implementation of VMsync.

In the current work, we measure changes to the memory and file-system images of a Xen virtual machine running the Android operating system. We drive the experiments with popular Android applications, and report how many bytes would be transferred for a range of policies. For example, when playing a YouTube video, we show that sending dirty memory pages transfers 3 times more data than sending only the bytes that actually changed in those pages. These measurement results constitute a modest research contribution, as we are not aware of previous measurements of VM-image changes using such mobile-specific workloads.

Overall, we conclude that VMsync is a feasible approach deserving of further research. Our measurements show that there are significant opportunities to save costs by choosing certain synchronization policies over others. At the same time, many questions remain to be answered before we know which policies are most appropriate in which situations.

2. RELATED WORK

System-level virtualization of mobile devices has been proposed and implemented by both researchers [5, 8, 3] and commercial entities [6, 12]. We agree with their conclusions that virtualization improves the security and manageability of mobile computing, and add our insight that virtualization would also enable seamless switching between devices.

There are several established techniques for migrating VM state between hardware hosts, but we find them unsuitable for our purposes. For example, *live migration* [4] transfers a VM image while the VM continues to run in the originating host, only suspending the VM for an imperceptible period while control is finally switched to the receiving host. However, live migration transfers the complete VM memory image each time, an operation that generally involves hundreds of megabytes if not gigabytes of data, which would be prohibitive over a slow wireless link. In addition, live migration assumes a high-speed shared storage medium between the hosts involved, so that file-system state need not be transferred at migration time. Mobile devices do not enjoy such high-speed shared storage.

A recent refinement on live migration uses delta compression to reduce the amount of data transferred in the later stages of migration [10]. However, it still sends the complete contents of memory at least once before beginning to apply differencing techniques to the pages that have changed in the course of the migration. It also still assumes a high-speed shared storage medium.

Work on *opportunistic replay* [2] proposes an approach for decreasing the amount of data transferred during VM migration in low-bandwidth environments. This approach logs user-input events (e.g., keyboard presses and mouse clicks) during VM execution, then transmits and replays this log on a second identically-configured VM to produce nearly the same VM state. Because only user events are logged, events

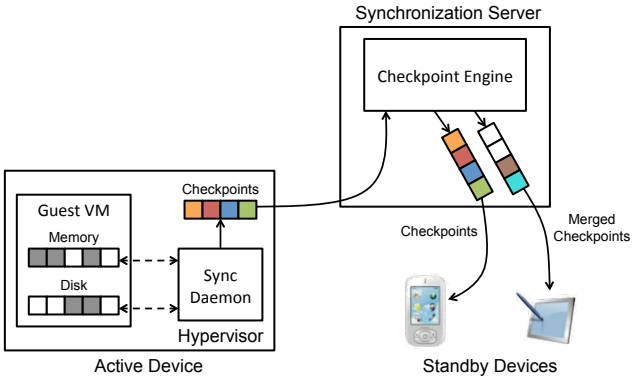


Figure 1: VMsync Architecture

triggered by additional hardware devices or background network connections may produce unmonitored state changes. These changes remaining after replay are also transferred and applied, resulting in a final identical VM. Though opportunistic replay is a potential mechanism for synchronizing multiple VMs in the VMsync scenario, there are many policy decisions left to explore to create a mobile VM synchronization solution that is imperceptible to the user. Further studies on opportunistic replay using today's network-intensive mobile applications and operating systems would also be needed to determine how well the approach would work in the VMsync scenario.

The Kimberley system [13] introduced the concept of pre-distributing a base VM image to relevant hosts, then sending only the differences from the base when wanting to move a VM from one host to another. It calls for suspending the VM on the originating host, calculating differences, transferring them, and applying them, before resuming the VM at the destination host. We adopt the idea of pre-distributing a base VM, but go further in pursuing incremental synchronization of live VM state among multiple devices without perceptibly suspending the VM.

3. PRELIMINARY DESIGN

The goal of VMsync is to maintain a consistent VM image across multiple devices while minimizing the time it takes for users to switch between devices. We consider this time the *switch penalty*. The simplest approach would perform a live migration of the guest VM at the time the user would like to switch devices, though this would incur a data transfer cost on the order of the VM size. For example, live migration of a Xen VM with 800 MB of memory can transfer as much as 960 MB (1.2x) [4].

In the case of wireless networks, VM state changes could occur faster than bandwidth allows, leading to higher switch penalties. Even with delta compression [10], live migration transfers on the order of the VM memory size during its initial stages, limiting its use in networks confined by data caps and bandwidth limitations. Therefore, along with reducing the switch penalty, we must also minimize the total amount of data transferred between devices. In VMsync we propose an incremental synchronization method to migrate a VM across mobile devices that attempts to minimize both the switch penalty and the data transferred.

Figure 1 represents the architecture of VMsync, a system made up of multiple host devices with virtualization support

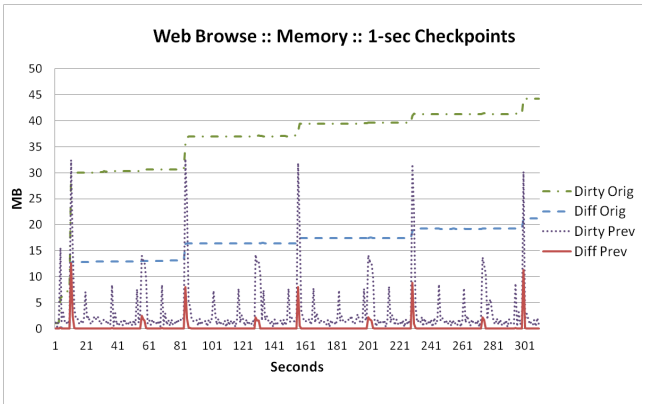


Figure 2: Memory contents change significantly with each new web page loaded.

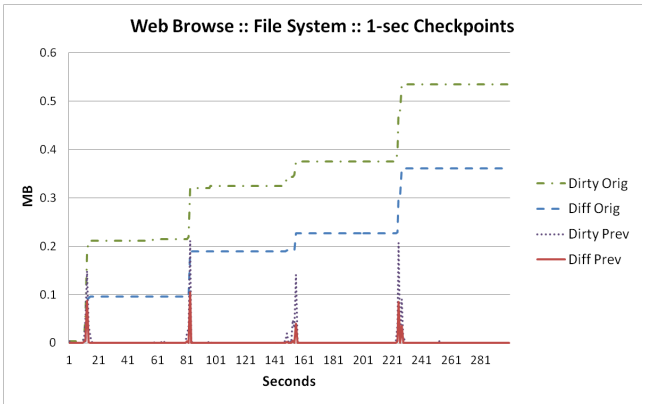


Figure 3: File-system contents change when the web browser caches data.

and a resource-rich server in the cloud, used as a synchronization point between devices. Devices registered with a VMsync instance are provisioned with a single base guest VM, like in Kimberley [13], containing a typical mobile device operating system such as Android. The hypervisor of each device handles syncing operations through a privileged daemon which monitors the guest VM state.

In our initial design of VMsync, only one *active VM* will be running at any given point in time to ensure that VMs do not diverge. This *active device* will propagate changes of both memory and file system state to the synchronization server over the period of time in which the device is active. We consider this process a *checkpoint*. Every other VM, considered *standby VMs*, will be *paused* and periodically updated via the synchronization server if the device is online. Devices which are not connected to the network or have not been synchronized will be considered to be in a *stale state* and must be synchronized before a user can switch to that device. The longer a device is offline or not updated, due to limited bandwidth or policy decisions, the higher the switch penalty.

The synchronization daemon running on the end device, which monitors the guest VM for changes, must be designed in such a way that balances the tradeoff between data transferred and computational overhead. For example, a naive low-computation approach would be similar to live migration, e.g., simply propagate every memory page and disk block that has been changed since the last checkpoint. During our feasibility study in the subsequent section, we show that this method would propagate a large amount of data that has not actually changed. This raises the question of how to efficiently synchronize only the bytes that have changed since the previous checkpoint.

Since mobile devices contain an increasing amount of file-system storage, 64 GB or more, it would be feasible to maintain a snapshot of the previous memory checkpoint on disk. Today's mobile devices, such as smart phones and tablets, contain a limited amount of memory, typically maxing out at around 2 GB. This would allow a byte by byte comparison or an on-device differencing algorithm to identify the bytes that have changed since the last checkpoint. Alternatively, a network-based differencing algorithm such as *rysnc* could be used, though this may require additional network and computational overhead. Due to the large size of the file system, a copy-on-write disk image or customized block driver could be used to efficiently monitor file-system changes and synchronize file-system state. Previous work on opportunistic replay [2] and delta compression [10] could also be adapted for use during VMsync's checkpointing step.

Though the above update mechanisms are not novel, VMsync introduces many policy questions that can only be answered with a thorough design, implementation, and evaluation of the system. For example, in order to minimize the switch penalty, should the active device propagate changes periodically over time, use specific operating system events to infer the best time to propagate changes, or use a mix of these two policies? An event, such as putting the phone to sleep via the hardware power button, may be a good indicator that the device will no longer be used for some time and the user could potentially switch to another device. On the other hand, if the user switches devices before this event, the state change from the last checkpoint may be large and will thus increase the switch penalty. In this case, a periodic checkpoint would have helped. We can also use other factors such as location, nearby device presence, battery life, CPU utilization, network bandwidth, bandwidth caps, etc., as triggers for state propagation.

There are also various policy decisions that must be made on the synchronization server. For example, when should devices be updated with the latest state information? One policy may decide that only devices connected to a network with sufficient bandwidth can receive changes. For devices that have been offline for some time, the server can merge multiple checkpoints from the active VM to minimize the amount of data transferred. In some cases it may also be possible to bypass the synchronization server by using local wireless links such as Wi-Fi Direct, Bluetooth, or NFC to migrate changes directly between devices.

Finally, the variety of hardware configurations in mobile devices introduces challenges when migrating a VM from one device to another. We have not fully addressed this device heterogeneity issue, but we note that it is common to other mobility schemes based on VM migration [7, 11]. On the positive side, modern mobile operating systems such as Android and Windows Phone 8 are designed for extensibility, thus providing support for many types of hardware built by different manufacturers. Therefore, it seems feasible in the future to extend these operating systems to detect and adapt to hardware changes at runtime.

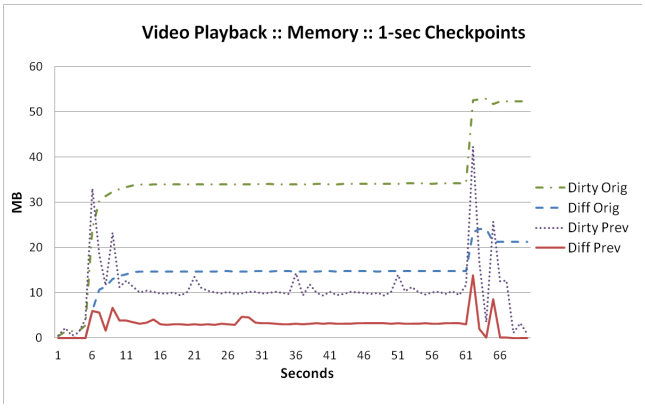


Figure 4: Most changes to memory occur during the initial loading of the video, with some final changes when the application closes.

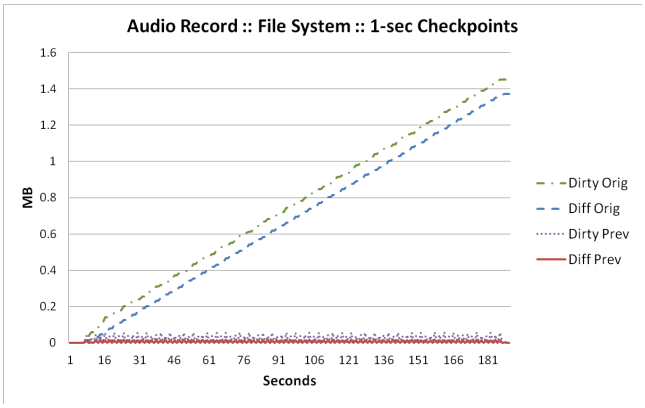


Figure 5: File-system contents change continuously during audio recording.

4. FEASIBILITY STUDY

To measure the feasibility of VMsync, we analyzed changes to memory and the file system under various mobile workloads. Our goal was to determine how much data would be required to maintain a consistent VM state across multiple devices. As workloads, we chose applications that we believe are representative of current mobile phone use: web browsing, video playback, audio playback, and audio recording.

We performed our study on the Android platform. Our VM is an Android-x86 4.0.4 (Ice Cream Sandwich) [1] guest domain running above the Xen 4.1.1 hypervisor. The Android VM uses an Android Open Source Project (AOSP) 3.0.8 kernel compiled for x86 and with Xen paravirtualization support enabled. The guest domain is allocated with 512 MB of memory, 1 virtual CPU, a 512 MB read-only system image that contains the Android software stack and is pre-distributed to all devices, and a 512 MB read/write data partition that is monitored for changes during experiments. The host machine is a desktop-class machine with a quad-core Intel Core i7 860 processor executing at 2.8 GHZ and 12 GB of RAM. Though this host system is in no way representative of a mobile device, we are measuring changes to memory and file-system usage under mobile workloads, not computation overheads or other effects influenced by host capacity.

Our synchronization daemon executes outside of the guest domain within Domain0, and uses xenctrl APIs to map the guest domain's memory prior to starting a workload. At the beginning of the workload, we pause the VM and save a copy of both memory and the data partition. This represents the original state of the VM prior to running a workload. Over the course of a workload, we analyze the memory and file system states across various *checkpoints*. For each checkpoint, we pause the VM and compare the current memory and file system state with the original copy we saved in the beginning of the workload. We also save the state of each previous checkpoint to measure changes over specified intervals of time. This procedure simulates an implementation of VMsync, where a device would periodically sync the current changes with all devices. We can thus understand the volume of data changed across an entire workload for different VMsync policies.

Measuring Checkpoint Sizes

In each of Figures 2–5, we show four curves, each corresponding to a different checkpointing policy. The top curve, labeled *Dirty Orig*, represents the case where each memory page or disk block that was changed from the beginning of the workload is synced. The second curve, labeled *Diff Orig*, only counts the bytes changed from the beginning of the workload, simulating a differencing syncing mechanism. The third and bottom curves, *Dirty Prev* and *Diff Prev*, follow similarly, but are measured with respect to the system state of the previous checkpoint. For example, Figures 2 and 3 shows the number of megabytes that would be transferred while executing a checkpoint every second during a web browsing session using the default Android browser.

Our web browsing workload navigates through a popular news article on m.cnn.com. The workload sleeps a predetermined number of seconds (45) before scrolling down the page as a normal user would. Then the workload sleeps again (25 seconds) to simulate reading the end of a page before moving to the next page of the article. Each time the browser loads a new web page, we see a significant change in both memory and file system activity. The changes in the file system are due to the fact that the Android browser maintains a cache of previously viewed web pages and thus there are additional changes each time a new page is loaded.

Figure 4 shows the checkpointing effects during a streaming video using the YouTube app for Android. In the case of YouTube, the application is launched and the video is buffered during the beginning of the workload. During video playback, many pages are modified but the overall change in the system remains steady. An important observation is that throughout all our workloads, the most significant overall change to memory (Diff Orig curve) also occurs at the beginning of the workload. After this initial change, the changes following are fairly minor. For this reason, if there is enough time to propagate the changes caused by starting an application, migrating to a different device later in the use of an application should only incur a small switch penalty. On the other hand, if a user switches to a second device within a few seconds of starting an application, the amount of time needed to sync the system state would be much longer and possibly noticeable by the user.

To observe the effects of a file-system intensive workload, we used the Hi-Q MP3 Voice Recorder app to record 3 min-

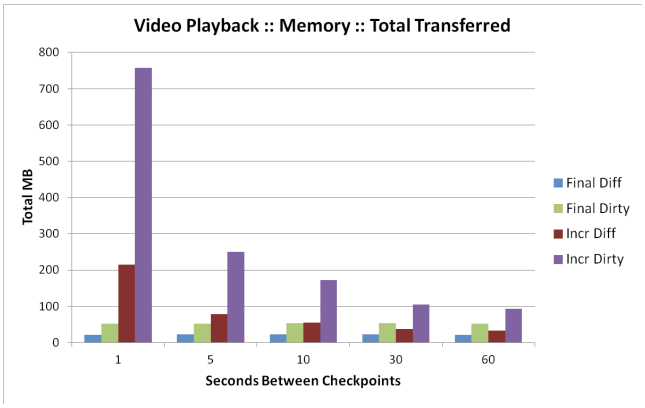


Figure 6: Transferring complete dirty memory pages involves three times more data than necessary.

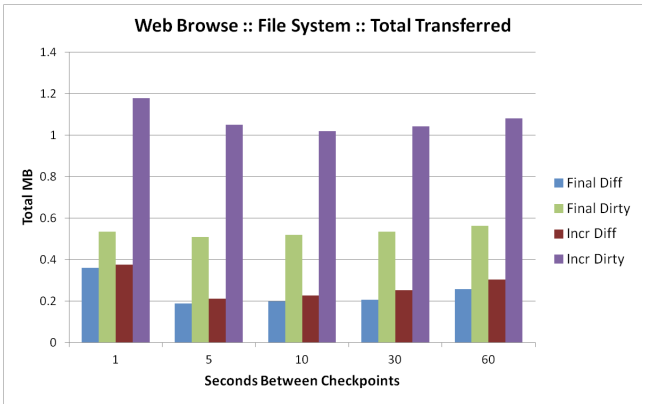


Figure 7: Transferring only the bytes that have changed is also advantageous in the file-system case.

utes of audio within our VM. This workload continuously writes to the file system and finishes with a final 1.3 MB change to the file system. Figure 5 shows the results when executing a file system checkpoint every 1 second. When compared to the original system image, the changes over the course of the workload are continuously increasing over time. Every checkpoint, the changes between the previous state are consistently minimal, but add up to the final total change at the end of the workload. Though a user may not switch devices while recording an audio session, this represents the feasibility of switching between devices when data is being constantly written to the file system.

Measuring the Total Bytes Required for Sync

For each workload, we vary the time between checkpoints from 1 second to 5, 10, 30, and 60 seconds. Due to the fact that each workload runs for a fixed amount of time, the number of checkpoints decreases as the amount of time between checkpoints increases. Because of this, the overall shape of each checkpoint curve is similar, but smoother and less fine grained over time. Figures 6, 7, 8, and 9 compare the total amount of data required for each checkpointing policy against the time interval between checkpoints. While we are not measuring the switch penalty directly in units of time, the size of data transfers under various policies gives us some indication of which policies would have higher or lower switch penalties. *Final Diff* represents the exact number of megabytes changed at the end of the workload, where *Final Dirty* is the total number of megabytes required if you synced each dirty page or block. When using a policy that compares against each previous checkpoint, we sum the number of megabytes changed for each checkpointing round and report the result as the total number of megabytes changed for that policy. *Incr Diff* represents the total number of bytes changed, where *Incr Dirty* represents the total number of complete pages that have changed over incremental checkpoints.

In all cases, the Final Diff and Final Dirty results remain constant (within a standard deviation of the workload) across varying time intervals. As the time interval between checkpoints increases, the number of checkpoints decreases, lowering the total number of megabytes for incremental checkpoints. Because memory state changes very frequently while a workload is running, checkpointing memory state often, such as every second, transfers large amounts of data. In the case of video playback, shown in Figure 6, propagating each memory page that has changed would require three times the amount of data compared to propagating the difference of each page. In fact, propagating dirty pages could use up to 35 times the amount of data than necessary in the worst case, measured from our audio recording workload. In all cases, propagating only the bytes that have changed at the end of the workload uses the least amount of data, but could cause a severe delay if the user decides to switch devices while running an application. It is also important to note that both differencing policies, final and incremental, may incur significant time and space overhead. We plan to measure these overheads and explore the trade off between each in a full-fledged implementation of VMsync.

Compared to memory state, the changes in file system state are fairly minimal, even in a file-system intensive workload such as audio recording (Figure 8). As the time between checkpoints increases, propagating dirty blocks every checkpoint converges closely to the actual number of megabytes changed. This is because during sequential writes, data is written to a single block at a time and the number of dirty blocks ends up being proportional to the total number of bytes changed. For writes spread out across multiple different blocks, such as in the web browsing workload in Figure 7, propagating dirty blocks incurs a cost about five times higher than performing a final diff.

Since mobile devices users can perform multiple tasks at once, we performed a combined workload which added background music, using the default Android music player, to our web browsing workload. In general, playing audio in the background has minimal effect on the system state. As shown in Figure 9, the size of changes to both the memory and file system state are nearly the same as web browsing alone, leading us to infer that audio playback continuously modifies a fixed number of memory pages and does not change the file system.

Following are some overall findings from this study. One, naive policies that only propagate dirty pages or blocks transfer large amounts of data and would not be feasible for mobile devices and networks. Two, large savings result from performing differences of previous checkpoints in order to propagate only the bytes that have changed. We also see that increasing the amount of time between checkpoints will save bytes but may require a higher switch latency if a user

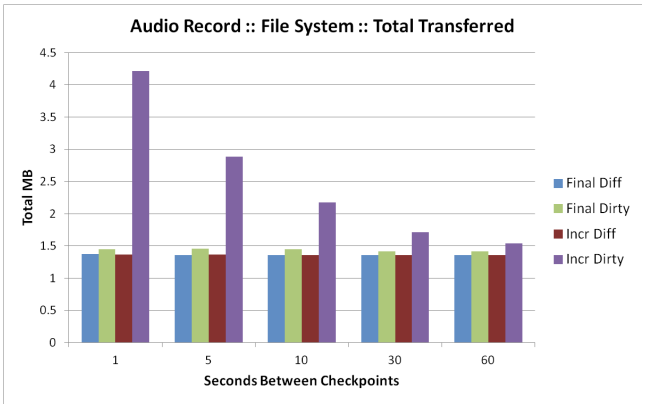


Figure 8: Waiting before propagating dirty file-system blocks approaches the cost of a differencing policy for apps that perform sequential writes.

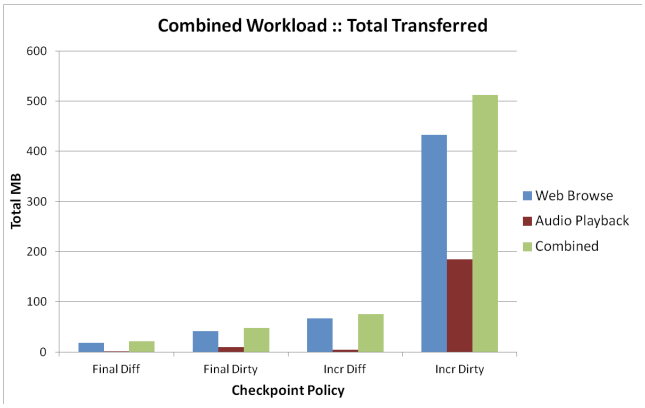


Figure 9: Playing music in the background of a web browsing session has minimal effect on the overall system state.

switches devices during that time. With regards to memory, the most significant changes occur in the beginning of a workload and subsequent changes are somewhat minimal.

5. CONCLUSION

We have presented our early work on VMsync, a system for incrementally synchronizing live virtual machine state among mobile devices. VMsync aims to enable users to seamlessly switch between devices with both data and computation state preserved across the switch without apparent delay. We described our initial design for identifying changes to the memory and file-system images of an active VM on one device, then propagating those changes to standby VMs on other devices via a synchronization server in the cloud. We also presented our measurements study of how much data would need to be transferred to maintain a consistent VM image across devices under different workloads and synchronization policies. From our efforts to date, we conclude that VMsync is a feasible approach with many open issues deserving of further research.

6. REFERENCES

[1] Android-x86 - Porting Android to x86. http://www.android-x86.org/.

[2] A. Surie and H. A. Lagar-Cavilla and E. de Lara and M. Satyanarayanan. Low-bandwidth VM Migration via Opportunistic Replay. In *Proc. 9th workshop on Mobile Computing Systems and Applications*, 2008.

[3] J. Andrus, C. Dall, A. V. Hof, O. Laadan, and J. Nieh. Cells: A Virtual Mobile Smartphone Architecture. In *Proc. 23rd Symposium on Operating Systems Principles*, 2011.

[4] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Proc. 2nd Symposium on Networked Systems Design & Implementation*, 2005.

[5] L. P. Cox and P. M. Chen. Pocket Hypervisors: Opportunities and Challenges. In *Proc. 8th IEEE Workshop on Mobile Computing Systems and Applications*, 2007.

[6] K. Gudeth, M. Pirretti, K. Hoeper, and R. Buskey. Short Paper: Delivering Secure Applications on Commercial Mobile Devices: The Case for Bare Metal Hypervisors. In *Proc. 1st ACM Workshop on Security and Privacy in Mobile Devices*, 2011.

[7] M. Kozuch and M. Satyanarayanan. Internet suspend/resume. In *Proc. 4th IEEE Workshop on Mobile Computing Systems and Applications*, 2002.

[8] M. Lange, S. Liebergeld, A. Lackorzynski, A. Warg, and M. Peter. L4Android: A Generic Operating System Framework for Secure Smartphones. In *Proc. ACM Workshop on Security and Privacy in Mobile Devices*, 2011.

[9] Mashable. App Store Stats: 400 Million Accounts, 650,000 Apps. `http://mashable.com/2012/06/11/wwdc-2012-app-store-stats/`, June 2012.

[10] P. Svärd and B. Hudzia and J. Tordsson and E. Elmroth. Evaluation of delta compression techniques for efficient live migration of large virtual machines. In *Proc. 7th ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments*, 2011.

[11] R. Cáceres and C. Carter and C. Narayanaswami and M. Raghunath. Reincarnating PCs with Portable SoulPads. In *Proc. 3rd International Conference on Mobile Systems, Applications, and Services*, 2005.

[12] VMware. Verizon Wireless and VMware Securely Mix the Professional and Personal Mobile Experience with Dual Persona Android Devices. `http://www.vmware.com/company/news/releases/vmw-vmworld-emea-verizon-joint-10-19-11.html`, October 2011.

[13] A. Wolbach, J. Harkes, S. Chellappa, and M. Satyanarayanan. Transient customization of mobile computing infrastructure. In *Proc. 1st Workshop on Virtualization in Mobile Computing*, 2008.

Enabling the Transition to the Mobile Web with WebSieve

Michael Butkiewicz*, Zhe Wu*, Shunan Li*, Pavithra Murali*
Vagelis Hristidis*, Harsha V. Madhyastha*, Vyas Sekar†
* University of California, Riverside † Stonybrook University

1 Introduction

Web access on mobile platforms already constitutes a significant (> 20%) share of web traffic [3]. Furthermore, this share is projected to even surpass access from laptops and desktops [11]. In conjunction with this growth, user expectations for the performance of mobile applications and websites is also growing rapidly [15]. Surveys show that 71% of users expect websites to load almost as quickly as their desktops and 33% of annoyed users are likely to go to a competitor's site leading to loss of ad- and click-based revenue streams [1].

However, the performance of the mobile web today is quite poor. Industry reports show that the median web page takes almost 11 seconds to load over 3G networks even on state-of-art devices such as iPhone5 and the Samsung Galaxy S3 [2]; LTE is only marginally better at improving the latency. The key challenge here is that, unlike traditional devices, mobile devices are fundamentally constrained in several ways in terms of networking, compute, and storage capabilities that can cause high page load times [27, 26].

We are far from being alone or the first to identify these trends. In fact, there has been renewed interest in optimizing web performance focused specifically on mobile devices as evidenced by the proliferation of: a) public measurement reports and repositories (e.g., [7]), b) new optimized protocols (e.g., [13]), c) startups that help providers to generate mobile-friendly web pages (e.g., [10]) and to increase mobile performance (e.g., [14]), d) proprietary optimizations (e.g., [4, 12]), and e) better browsers (e.g., [24, 28]).

Despite the growing realization and recognition of these issues, surveys shows that over 90% of websites are not mobile friendly today [8]. We speculate that this disconnect between the need to customize for mobile devices and the actual adoption of proposed solutions stems from two related factors. First, mobile-specific customization seems to be expensive and often involves manual intervention, thereby restricting its adoption only to high-end website providers. For example, the fraction of websites with mobile-optimized versions drops from 35% in the top 200 to 15% among the top 2000.

The second, more fundamental, issue is that, the desire to deliver rich services (and associated ads and analytics) has, over the last few years, dramatically increased the complexity of websites; rendering a single web page involves fetching several objects with varying characteristics from multiple servers under different administrative domains [16]. This complexity leads to poor interactions with mobile-specific constraints due to several factors such as the need to spawn many connections, high RTTs on wireless links, and time to download large objects on low-bandwidth links. Furthermore, this is accompanied by a corresponding increase in the complexity of website generation (especially for dynamic content); thus, re-architecting them for mobile-friendly designs would require complete overhauls or parallel workflows, further moving the mobile web out of the reach of low-end website providers.

Our overarching vision is to democratize the ability to generate mobile friendly websites, enabling even small web providers to support mobile devices without investing significant resources to do so. While others have focused on automatically adapting web page layouts for mobile devices [17] and on optimizing the load times of Javascript-heavy websites [22], our focus is on reducing the high load times seen on mobile devices for generic web pages. Given the concerns surrounding website complexity and the need to avoid overhauling existing content management workflows, we take a pragmatic approach and cast the goal of customizing websites for mobile devices as an *utility maximization* problem. Specifically, we can view this as a problem of selecting a subset of *high utility* objects from the *original* website that can be rendered within some load time budget for user tolerance (say 2–5 seconds [18, 19]). We can then either block or de-prioritize the loads of low utility objects to reduce user-perceived page load times [9].

While this approach sounds intuitively appealing, there are three high-level requirements that need to be addressed before the benefits can be realized in practice:

- *Structure-awareness:* Given the complex inter-dependencies between objects in most web pages today, blocking or delaying the load of one object may result in several other dependent objects also being filtered out or delayed, e.g., if a Javascript is not fetched, neither will any of the images that the script would have fetched. Thus, even though an object may not directly contribute to the user experience (not visible to users), it may be critical for downloading useful content.
- *Utility-awareness:* Indiscriminate filtering of objects from a web page may prune out content critical to the website's functionality and render the mobile version of the web page useless. We need mechanisms for predicting the expected utility that an user gets from different objects on a given web page. Two concerns arise: (1) we may not know the utility a user perceives in advance before actually downloading the object, and (2) users may differ in their preferences, e.g., some users may dislike ads and images but others may perceive value in these.
- *Practical optimization:* Object selection problems to maximize some utility subject to budget/dependency constraints are typically NP-hard. Additionally, due to the complex policies involved in how browsers parallelize the loading of objects on a web page, it is hard to estimate the resulting page load time when a particular subset of objects on a web page are loaded.

Corresponding to each of these requirements, we describe key practical challenges and preliminary results from our current efforts in designing a *WebSieve* prototype: (1) Naive solutions for depen-

ACM HotMobile'13, February 26–27, 2013, Jekyll Island, Georgia, USA.

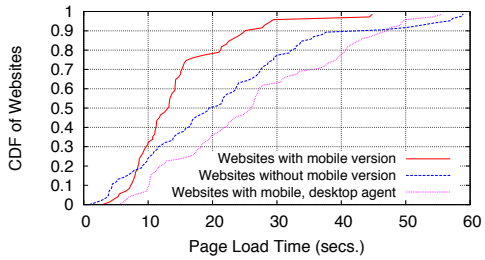


Figure 1: *Page load times on sites with/without mobile versions.*

dency extraction are unlikely to work in face of dynamic content and needs systematic solutions to extract causal relationships, but a practical "block-and-infer" strategy appears promising (§4); (2) We report experiences from a user study suggesting that any framework for assigning utilities to objects needs to account for user-specific preferences (§5); and (3) Despite the theoretical intractability, we can find practical near-optimal solutions in conjunction with approximate load time estimates (§6).

We do acknowledge that blocking low utility objects to reduce page load times may affect the page's functionality; e.g., a button may not be functional if an associated Javascript has not been loaded or the layout may be undesirable if the CSS has not been loaded. The main challenge here is the need to automatically capture the complex inter-dependencies that exist on today's web pages. While we discuss potential approaches here to reduce the likelihood of breaking web page functionality, striking the right balance between load time, utility, and functionality forms the crux of our ongoing work.

2 Motivation

Opportunity to reduce load times: We consider a dataset of 2000 websites from Quantcast's list of the top million websites—400 each, chosen at random, from the rank ranges 1–400, 400–1000, 2000–2500, 5000–10000, and 10000–20000. We identify which of these websites have mobile-optimized web pages. Figure 1 compares the load times [1] on the Sony Xperia smartphone for randomly chosen subsets of 100 websites that have mobile versions and 100 websites that do not currently have mobile versions. (The measurements were made over a 3G connection in a residential location at Riverside.) First, we see that the sites that have mobile versions have significantly lower load times compared to those do not. Second, the load time distribution for websites that do not have mobile versions is comparable to those for the normal/desktop version for the websites that have mobile-optimized versions. In other words, these unoptimized sites have not intentionally chosen to avoid customizing their websites because their load times are already low—there is significant room for reducing the load times. Third, 60% of the mobile-optimized websites still take more than 10 seconds to load, suggesting that even these could benefit from our proposed optimizations. *These results show that there is significant opportunity for reducing the load times of web pages on mobile devices.*

Website complexity causes high load times: A key contributing factor to high page load times is the increasing complexity of web pages. Our recent work [16] showed that, on average, loading a web page requires the browser to fetch over 50 objects from more than 10 servers. Such complexity is not restricted to top-ranked websites, but it exists across web pages in all rank ranges—even among sites in the 10000 to 20000 rank range. In fact, our prior work showed that the number of objects on a page is the most correlated with load time [16, 27]. *Therefore, in order to reduce page load times on smartphones, a key step is to have a systematic solution to "tame" this web page complexity.*

[1] A page's load time is the time at which the `onLoad` event is fired when the page is loaded on the default Android browser.

Version	% responses citing significant loss of useful information			
	Set1	*Set2*	*Set3*	**Aggregate**
Flashblock	20	20	20	**20**
NoScript	0	20	70	**40**

Table 1: *User study to quantify usability impact of naive customization techniques. Numbers are reported with one significant digit given dataset's size.*

Naive approaches to tame complexity do not work: To reduce the impact of a web page's complexity on page load times, we need to either load only a subset of the objects on the page or prioritize the loads of "important" objects. A strawman solution is to filter all objects of a particular type that users may consider to be of low utility. For example, we can use browser extensions such as *Flashblock* and *NoScript* to block all flash and script objects, and all other objects that these cause to be loaded. To analyze how well this would work, we ran a preliminary user study over several websites. We chose three subsets of 10 websites from the top 1000 websites ranked by Quantcast—*Set1* comprised 10 websites chosen at random, *Set2* was the top 10 sites based on the number of unique origins contacted, and *Set3* consists of 10 randomly chosen mobile-optimized websites.

We conducted a survey across 33 participants by hosting the website `http://website-comparison.appspot.com`. We asked users to compare a screenshot of the default version of these 30 sites with screenshots for two alternatives—those obtained with the *Flashblock* or the *NoScript* extension enabled (We use the extensions with their default configurations.) Table 1 shows that the use of either *Flashblock* or *NoScript* would significantly impact user experience. While users may not consider scripts included in a web page as important, blocking those scripts impacts user experience since the objects fetched by executing those scripts are blocked as well. *Thus, to block or de-prioritize low utility content on a web page, we need to take into account the role played by every object on that page as well as the dependencies between objects.*

On the other hand, though ads and objects related to analytics may not be critical to the user experience, blocking these objects can be detrimental to the interests of website providers. *Therefore, reducing web page complexity by blocking objects also needs to take into account the implications of doing so on provider interests, e.g., the impact on their revenue.*

3 Vision and Roadmap

Next, we present a high-level overview of our envisioned WebSieve architecture to instantiate the approach of reducing web page complexity to reduce page load times on mobile devices. Our focus here is primarily to achieve the right balance between page load time, user experience, and website providers' interests. We do not focus on orthogonal problems in optimizing web pages for mobile devices such as customizing the page's layout to suit the screen size, form factor, and UI capabilities of the devices [17].

The problem statement that lies at the heart of WebSieve is as follows. Given a budget on load time, our goal is to select a subset of objects on a web page that will maximize the user experience while satisfying the load time constraint. To keep the discussion simple, consider for now that (1) there is only one type of client device, and (2) that the website's content does not significantly change over time; we discuss how to address these issues in practice in Section 7. Consider a web page W that has a set of objects O. Each

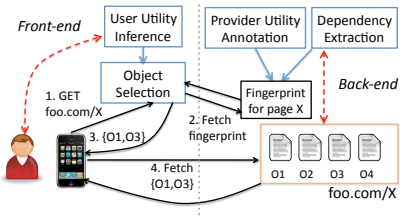


Figure 2: *Overview of WebSieve architecture.*

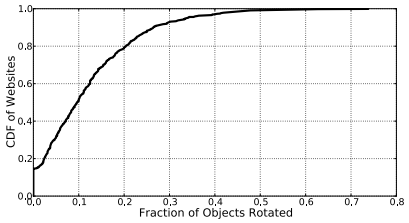


Figure 3: *Quantifying change in objects across page loads.*

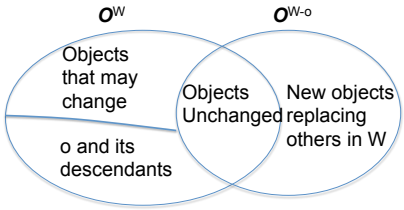


Figure 4: *Visualizing our intuition for mapping objects.*

object $o_i \in O$ takes time t_i to fetch from the server and offers *utility* $Util_i$ to users and website providers. Given a maximum allowed load time of M (e.g., user studies suggest a tolerance around 2–5 seconds [19]), our goal is to select, from all subsets of objects $O' \subseteq O$ whose load time is less than the allowed maximum (i.e., $\leq M$), the subset that maximizes the total utility $\sum_{o_i \in O'} Util_i$. WebSieve can then reduce the user-perceived load time for page W either by loading only the selected subset of objects or by loading this subset before other objects on the page.

This abstract view of the problem highlights three key design challenges that we need to address.

- **Dependencies:** There are natural loading dependencies between the objects in a web page. For example, many web pages download images as a result of executing Javascripts on the client; in this case, the script is a natural *parent* of the resultant image. This means that we cannot select to load an object without choosing to load its parent as well.
- **Utility inference:** The notion of utility perceived by users and providers is a complex issue. For instance, objects in a web page that are not "visible" may not directly contribute to the user experience but are indirectly critical to download interesting content. Moreover, users may differ in their interest and tolerance to web objects, e.g., some users may hate ads and images but others may perceive value in these.
- **Object selection:** The algorithm to select a subset of objects needs to be very efficient; otherwise the runtime of the algorithm may be better spent loading more objects. A key challenge here is that predicting the load time when a subset of objects is fetched is itself non-trivial. In addition to the parent-child relationships described above, web browsers employ parallelization techniques to accelerate page loads. For example, a default Firefox desktop installation can maintain up to 16 connections in parallel and at most 6 connections open to any particular hostname. Consequently, the time to load a set of objects cannot be simply modeled as a simple combination of the individual load times. Furthermore, the load time also depends on the specific device and operating conditions, e.g., 3G vs. WiFi connection.

Figure 2 depicts how these three components fit into WebSieve's architecture, which can be logically partitioned into a frontend and a backend. For every web page, the backend generates a compact *fingerprint* that summarizes the key properties of the page. This fingerprint includes a) the dependency structure of the web page, b) load time information for objects on the page, and c) the object utilities as perceived by the website provider. Since these features change infrequently, WebSieve's fingerprint generation can be performed offline. A website provider can host the backend for fingerprint generation of all pages on his site, or this task can be deferred to any third-party server-side infrastructure. The frontend, which customizes web pages on the fly, can be implemented either as a browser extension or in a proxy that supports dynamic page rewriting and Javascript execution capabilities.

The typical steps involved in fetching and rendering a page with a WebSieve-enabled client will be as follows. The client requests the base HTML file for the web page via the frontend. Along with the HTML file, the frontend also fetches in parallel the fingerprint for the web page from the backend. By combining this fingerprint with the utilities expressed by the local user, the frontend determines which subset of objects on the page it should load. When the client's browser issues subsequent requests for the remaining objects on the page via the frontend, the frontend either sends an empty response or defers the load for objects that are not in its selected subset.

4 Dependency Extraction

As websites increasingly contain dynamic content loaded by scripts and customizable widgets, a specific object can be identified and downloaded only after its logical parents have already been processed. Consequently, any attempt at choosing a high-value subset of objects must account for these logical dependencies. Our goal is to automatically infer the load dependency graph for any given web page; manual specification of cross-resource dependencies by the web page's developer is impractical since 30% of the objects on the median web page are fetched from third-party domains [16].

4.1 Strawman solutions

Consider a page W consisting of the set of objects $O^W = \{o_1 \ldots o_n\}$. Each object o_i has a logical dependency on its *parent* p_i. We consider two intuitive solutions to infer this parent-child dependency structure within a web page. Note that these dependencies cannot be directly inferred from the document structure of the page, as some objects may be loaded after executing dynamic scripts.

HTTP Referer: The first option is to rely on HTTP *referer* tags to identify the causal relationships between object loads. We can load the web page once, and then extract the referer tags during this load to infer parent-child relationships. While this seems intuitively simple, this is not robust. A dominant fraction of dynamic content is loaded via Javascript which does not yield useful referer tags.

Structure inference via blocking: The high-level idea here is to infer potential causal relationships by blocking objects in the web page, similar to the idea proposed in WebProphet [21]. Suppose that the set of objects on a page do not change across multiple loads. For each object $o_i \in O^W$, we can load the webpage when this object is blocked; we perform these page loads with individual objects blocked on a server using the Firefox browser (in its default configuration) with an empty cache. If the page observed when blocking o_i is W_{-o_i}, then we know that every object in the *set difference* between the object sets $O^W - O^{W-o_i}$ is a logical descendant of this blocked object o_i. Though this one step may not be able to distinguish between immediate descendants and indirect descendants, by repeating this experiment for every object, we can reconstruct the exact dependency graph between objects.

In practice, however, web pages are not static even within a short window of time; the set of objects loaded across back-to-back loads

of the same page can be different, e.g., because every refresh yields a different ad or causes the website provider to return a different banner image. For example, in our dataset of 2000 websites, Figure 3 shows that over 10% of objects on the page change across page refreshes in 40% of sites. In our context, this implies that the set of objects loaded after blocking o_i will not be a strict subset of the original set of objects O^W. Specifically, some object from O^W could be missing from W_{-o_i} either because it is a logical descendant of o_i or because it was replaced with a different object when we reloaded the page (see Figure 4). Because of this ambiguity, we may potentially infer false dependencies (i.e., claim x is a parent of y, even though they are unrelated) using the above approach.

4.2 Proposed approach

To handle the constant flux in a web page's content, we propose the following approach. As before, let O^W be the set of objects in the original webpage and the set of objects seen after blocking o_i be O^{W-o_i}. At a high-level, we want to distinguish between the objects that are genuinely missing (i.e., descendants) vs. objects that have been replaced.

As a simplifying assumption, we assume that the number of objects in the web page does not change over the period of time it takes to infer the page's dependency structure; our preliminary measurements confirm that this is indeed the case. Then, we try to infer a *one-to-one* mapping between the set of objects in $O^{W-o_i} - O^W$ and $O^W - O^{W-o_i}$; note that this cannot be a bijection since the sizes of the two sets are different. The intuition behind our approach is that, when we reload the page after blocking o_i, some of the objects in the original web page have been subsequently replaced by the content provider. These new objects are the ones in $O^{W-o_i} - O^W$. Our goal then is to *match* each such object with a corresponding object in the original web page (i.e., without any blocking). Once we have this matching, we know the true set of "missing" objects as the ones that appear in $O^W - O^{W-o_i}$ but do not match up with any object in $O^{W-o_i} - O^W$. These are the true descendants of o_i.

We infer this correspondence between blocked objects and objects in the original web page with a two-step approach. The first step is to find where the object appears in the source files downloaded and match with the object that originally appeared in its place. In our measurements, we observe that this simple mapping step is able to accurately match over 80% of objects that change across page loads. Some objects remain unmapped after this step, for example, because their URLs are generated algorithmically by an embedded script. To address such cases, we map objects using object attributes (e.g., file type and file size) with a simple nearest neighbor like algorithm. With this two-stage approach, we obtain a comprehensive procedure for mapping objects across page loads.

5 Utility Inference

Next, we focus on inferring the utility of individual objects in a webpage. First, we consider the user-perceived utility of different web objects. Ideally, for every object on a web page, we want to run controlled user studies across a sufficiently large sample of users to evaluate the expected value that users perceive from that object.

Since it infeasible to do so for every single web object, we explore the possibility of learning a classifier that can estimate utilities. Though the number of objects on the Web is potentially unbounded and growing, the utility of any object will likely depend on a few important *characteristic features* of that object. For example, some of the candidate features may include attributes such as the location of the object on the web page (e.g., providers are likely to place interesting objects on top), the type of object (e.g., advertisement vs. image), whether the object has a clickable link, whether the object is visible on the page or hidden, and so on.

Our goal is to learn a *predictive model* that takes as input such object attributes and estimate the potential utility. More formally, if we have features $F_1 \ldots F_j \ldots$ (e.g., location, type) and we have an object where the values of the features are $\langle F_1 = f_1^i, f_2^i \ldots f_j^i \ldots \rangle$ (e.g., location=top-left, bottom-right) [25], the prediction model $Util(\{f_j^i\})$ takes as input the values of these features for a particular object and outputs the object's utility score.

User study to infer utilities: To gain initial insights into the feasibility of inferring such a predictive model, we ran a *user study* using the website `http://object-study.appspot.com`. On this site, we show every visitor snapshots of 15 web pages—the landing pages of 15 websites chosen at random from our list of 2000 sites (see Section 2). For each of these 15 web pages, we pick one object on the page at random and ask the user: *Would removing the 'Object of Interest' greatly impact a user's experience on the website?* We ask users to report the perceived "value" of each object on a Likert scale from -2 to 2, which correspond to an answer varying from "Strong No" to "Strong Yes" in response to our question. We collect responses to this survey from 120 users on Amazon Mechanical Turk.[2] An examination of the responses from our user study shows that simple heuristics such as categorizing all objects "below the fold" as low utility do not work; irrespective of where we consider the fold within the range of 600 to 900 pixels, we find that roughly 35% of objects below the fold were marked as important.

Need for personalization: We use the responses from our user study to train several types of classifiers (with five-fold cross validation) such as decision tree, SVM, and linear regression. Each sample of the training data comprises the features associated with a particular object as the attributes and a user's utility for that object as the value. We associate every object with various features that capture its size, its type of content, if it is an image, whether it is part of a sprite, the object's location on the page, whether it is visible and if so, whether it is in the foreground or the background of the web page, and if the object has a link, whether that link points to third-party content. However, we find that none of these features are well correlated with the responses in our user study; as a result, the best prediction accuracy that we were able to obtain is 62%.

Surprised by the low accuracy across all classifiers, we analyzed the responses. Specifically, we looked at different types of object features, and for each, we analyzed the user responses within that specific feature (e.g., button, ad, location). It became immediately evident that the problem was that we were trying to build a global model across *all users*. We observed that there is considerable variability in user responses within each feature. For instance, 20% of users felt background images were important while 60% did not, while only 50% of users thought links on the bottom were important. What was striking, however, was that any given user was *consistent* in her responses. That is, across all websites, a user typically rates over 80% of objects with a given feature as either important or unimportant.

Hence, we foresee the need for *personalization* in WebSieve. In other words, WebSieve needs to learn and use a classifier customized for a specific user. For example, after a user first installs the WebSieve frontend, the user can mark selected objects on any web page she visits as low utility (say, whenever the page takes too long to load); the Adblock browser extension similarly lets users mark ads that the user wants it to block in the future. Based on

[2] As a sanity check, we only pick respondents who pass our validation stage where we show 4 objects known to be extremely relevant or irrelevant and filter out users who respond incorrectly.

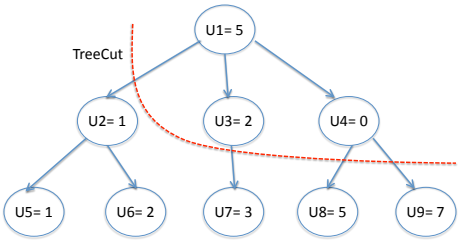


Figure 5: *Each arrow represents a logical parent-child dependency. We want to pick a subset of objects respecting the dependencies that maximizes the utility given a bound on load time.*

the user's responses, we can then learn a classifier over time that is specific to the user. However, a particular user's utility of objects with similar features may vary significantly across different types of websites. For example, small objects with links are likely to be important on a shopping website (e.g., the "shopping cart" button) but not as important on news sites (e.g., the *Like* and *+1* buttons). Therefore, WebSieve may need to consider different categories of websites, and even for a specific user, train a different classifier for each website category. A natural question here is the trade-off between increased accuracy of inferred utilities and overhead for the user as we need larger training sets.

Accounting for functional dependencies: If functional dependencies between objects are not accounted for, blocking objects can potentially break the web page's functionality and re-ordering object loads may not reduce user-perceived page load times even if high utility objects are loaded upfront. For example, delaying the load of a CSS object until after other high utility objects may result in a flash of unstyled content (FOUC) [6]. Similarly, if the Javascript that has registered an event listener with a button is not loaded, that button may not be functional. These are dependencies that our proposed approach in Section 4 will fail to detect. Hence, we directly account for such functional dependencies by associating CSS objects and Javascripts that have event listeners (which we conservatively detect via static analysis of Javascript code) with the highest utility. Based on our previous measurements [16], we estimate that these objects typically account for a small fraction of the objects on a web page.

Accounting for provider utilities: In addition to accounting for user-perceived utilities, it is important to ensure that the interests of website providers are preserved. Prioritizing only popular/useful content can hurt business interests of web providers because analytics or ads may get filtered out. To take this into account, WebSieve can allow for web providers to specify in any web page's source code the objects that are considered important by the provider of that page. For example, these prioritization hints can be added via `META` HTML tags. WebSieve can then take these priorities into account in combination with its estimates for user-perceived utilities. Thus, WebSieve can ensure that the interests of web providers are respected, while minimizing the burden on them for customizing their web pages for mobile devices.

6 Optimal object selection

We describe the abstract formulation of the object selection problem to highlight the key parameters involved, discuss practical challenges, and present our roadmap to address these.

6.1 Problem Formulation

The object selection module in the frontend receives the *fingerprint* from the backend which captures the dependency structure (§4) and annotations to specify key object features (§5). Using these features in conjunction with the user's preferences, it can compute the expected utility that each object provides. Combining these, it constructs a logical representation of the webpage as a *tree* where each node in the tree is annotated with its utility, as shown in Figure 5.

Our goal is to select a suitable *tree cut* in this tree structure; i.e. a cut that also satisfies the dependency constraints. Formally, we are given as input the page tree dependency T for a website W and the time budget M. If C denotes a cut, we want to select the cut C^* that, out of all cuts that can be loaded within time M, maximizes the expected utility.

It is evident that we need a fast algorithm that can solve this problem because object selection is on the critical path for loading the webpage. If the optimization itself takes too much time, then it defeats the purpose of reducing the page load time.

6.2 Practical Challenges

There are two key stumbling blocks. First, the dependencies between objects make this problem NP-hard.[3] Second, any optimization framework will need to model the time to load arbitrary subsets of objects. It is difficult enough to model the $LoadTime(C)$ function even for a specific fixed subset of objects, let alone for all possible subsets! This challenge arises from browser optimizations and the use of parallel connections in loading a web page. In particular, it is challenging to find a closed form function for $LoadTime(C)$. For example, some intuitive solutions like using the sum of the load times or dividing this sum by the expected number of parallel connections turn out to have very high ($\approx$ 3–4 seconds) estimation errors. Thus, we have a chicken-or-egg problem here—in order to pick the optimal subset we need to estimate the load time, but we cannot estimate this before picking a specific subset. In other words, without explicitly enumerating all possible subsets and physically loading them, it appears we cannot solve this optimization.

6.3 Proposed Approach

Dependency Modeling: To address the first problem of dependencies, we propose to use compact integer linear programming formulations. Let d_i be a $\{0, 1\}$ variable that indicates if we have selected the object o_i. Let p_i denote the logical parent of the object o_i in the page tree. Then the dependencies become a simple linear constraint of the form: $\forall i : \ d_i \leq d_{p_i}$.

Load time approximation: We see two practical approaches to address the load time estimation challenge. The key idea in both cases is to leverage the load time "waterfall" for the original web page annotated with the finish time t_i^f for each object i. This information can be included in the web page's fingerprint.

The first approach is to obtain a *conservative* load time estimate. Specifically, given a set of objects O, we can use the *maximum* finish time: $LoadTime(O) = \max_{i \in O} t_i^f$. This is conservative because blocking some objects will have scheduled this max-finish-time object much earlier. Given this context, we can write the page tree cut as an compact integer linear program (ILP). We do not show the full formulation due to space constraints. While we are still solving a NP-hard discrete optimization problem, we can leverage efficient solvers such as CPLEX. We find that it takes $\leq$ 30ms to solve the optimization with real dependency graphs for pages with $\approx$ 100 objects (but with synthetic utilities). Thus, despite the theoretical intractability, we have reasons to be optimistic.

The second, is to heuristically estimate the load time for a given subset of objects by using the timeline of object loads. The main idea is to look for "holes" in the waterfall after blocking and move

[3]We can formally prove via a reduction from the weighted knapsack problem, but do not present the reduction here for brevity.

all objects whose parents have already been loaded to occupy these holes greedily. While this does not give a closed form equation, it gives us a practical handle on estimating the load time for a subset, and we find it works well ($< 20\%$ error). With this estimator tool, we can use greedy "packing" algorithms; iteratively pick the object with highest utility and select it along with its ancestors as long as this choice does not violate the time budget.

We can also combine these two approaches to improve the optimality. For example, we can first run the ILP and then use the greedy approach to exploit the residual time left because of the conservativeness of the max-estimator. A natural concern is how close to the *optimal* solution our conservative ILP and greedy solutions are. In particular, we need to come up with mechanisms for getting tight upper bounds on the optimal solution given that the problem is intractable. We plan to investigate these in future work.

7 Discussion

Website stability: A web page's fingerprint needs to be regenerated as the set of objects on the web page and the structure of the web page changes. To gauge how often this regeneration of a web page's fingerprint will be necessary, we performed a preliminary study with 500 websites (chosen at random from our dataset of 2000 websites). We loaded the landing page of each of these websites once every six hours for a week. Our analysis seems to indicate that, though we saw previously that a significant fraction of objects on a web page change across repeated loads, the subset of stable objects and the dependencies between them appear to persist for several days. Hence, it will likely suffice for WebSieve to regenerate dependency information once a week. In practice, we can consider an adaptive scheme based on information provided by the website provider—refresh information more (less) frequently for web pages that have more (less) flux in their content.

Extrapolating across clients: Apart from dependency information, a web page's fingerprint also includes object load time information. The load times for individual objects however depend on client device capabilities, e.g., mobile phones vs. tablets or different versions of smartphone OS. For example, recent studies show that page load times significantly vary based on the device's storage [20]. Since it is impractical to gather load time information for every web page on every type of client device in every possible network condition, we need the ability to *extrapolate* load time across clients. This algorithm should take two inputs: 1) load time measurements on a reference device type in a specific network setting, and 2) a characterization of a target device and its network. Given these inputs, the algorithm should extrapolate measured load times to the setting of the target device. At the time of loading a web page, WebSieve's frontend can then use this algorithm to appropriately tailor load time information included in the web page's fingerprint for the local client.

Balancing user-provider utilities: One obvious issue here is the tension between users and providers; e.g., users may not want ads but providers do. Note that this problem is not intrinsic to WebSieve and exists today with Adblock-like solutions [5] and tracking [23]. While we cannot speculate how this tussle will play out, our utility maximization framework provides a technical solution to deal with this tussle more explicitly, in contrast to today's binary measures that pick extreme points catering to only one side.

Other applications: While we have focused here on the problem of reducing web page load times on mobile devices, our approach also has other applications. For example, blocking low utility objects can reduce the energy consumption associated with web browsing on mobile devices. Similarly, blocking low utility objects can help users of mobile devices cope with ISP-imposed caps on the amount of data they can receive over the network.

8 Conclusions

Our common mode of access to the Web is slowly transitioning from desktops/laptops connected to wired networks to mobile devices connected with access to wireless networks. While this client-side revolution is already under way, the ability to cope with this change is currently restricted to the top websites.

Our overarching vision is to democratize the ability to generate mobile-friendly websites, enabling even small web providers to transition to support mobile devices without investing significant resources to do so. For this, we present the WebSieve architecture, whose design is motivated by the observation that the Web performance problems on mobile devices stem from the increasing complexity of websites. To tame this complexity, the WebSieve architecture takes into account the intrinsic dependency structure of webpages and user-perceived utilities, and uses these to optimally select a subset of objects to render on mobile devices given a budget on the load time. While we have highlighted several open issues that need to be addressed to translate our vision into reality, our early approaches and results give us reasons to be hopeful.

References

[1] Google Research: No Mobile Site = Lost Customers. http://www.forbes.com/sites/roberthof/2012/09/25/google-research-no-mobile-site-lost-customers/.

[2] 2012 state of mobile ecommerce performance. http://www.strangeloopnetworks.com/resources/research/state-of-mobile-ecommerce-performance.

[3] 25 percent use smartphones, not computers, for majority of Web surfing. http://www.technolog.msnbc.msn.com/technology/technolog/25-percent-use-smartphones-not-computers-majority-web-surfing-122259.

[4] Amazon Silk. http://amazonsilk.wordpress.com/.

[5] Firefox adblock foe calls for mozilla boycott. http://www.informationweek.com/firefox-adblock-foe-calls-for-mozilla-bo/201805865.

[6] Flash of unstyled content (fouc). http://bluerobot.com/web/css/fouc.asp/.

[7] HTTP archive beta. http://httparchive.org/.

[8] Less than 10% of the web in 2012 is mobile ready. http://searchengineland.com/less-than-10-of-the-web-in-2012-is-mobile-ready-112101.

[9] Minimize initial display time to improve perceived web page speed. http://answers.oreilly.com/topic/498-minimize-initial-display-time-to-improve-perceived-web-page-speed/.

[10] Mobify. http://mobify.me.

[11] Mobile Internet will soon overtake fixed Internet. http://gigaom.com/2010/04/12/mary-meeker-mobile-internet-will-soon-overtake-fixed-internet/.

[12] Opera Mini & Opera Mobile browsers. http://www.opera.com/mobile/.

[13] SPDY: An experimental protocol for a faster web. http://www.chromium.org/spdy.

[14] Strangeloop: Speed up your website. http://www.strangeloopnetworks.com.

[15] Survey Report: What Users Want From Mobile. http://www.gomez.com/resources/whitepapers/survey-report-what-users-want-from-mobile/.

[16] M. Butkiewicz, H. V. Madhyastha, and V. Sekar. Understanding website complexity: Measurements, metrics, and implications. In *IMC*, 2011.

[17] Y. Chen, W.-Y. Ma, and H.-J. Zhang. Detecting web page structure for adaptive viewing on small form factor devices. In *WWW*, 2003.

[18] D. Galletta, R. Henry, S. McCoy, and P. Polak. Web Site Delays: How Tolerant are Users? *Journal of the Association for Information Systems*, 2004.

[19] F. Nah. A study on tolerable waiting time: How long are Web users willing to wait? *Behaviour & Information Technology*, 23(3), May 2004.

[20] H. Kim, N. Agrawal, and C. Ungureanu. Revisiting storage for smartphones. In *FAST*, 2012.

[21] Z. Li, M. Zhang, Z. Zhu, Y. Chen, A. Greenberg, and Y.-M. Wang. WebProphet: Automating performance prediction for web services. In *NSDI*, 2010.

[22] B. Livshits and E. Kiciman. Doloto: Code splitting for network-bound web 2.0 applications. In *FSE*, 2008.

[23] J. Mayer, A. Narayanan, and S. Stamm. Do not track: A universal third-party web tracking opt out. http://datatracker.ietf.org/doc/draft-mayer-do-not-track/.

[24] L. Meyerovich and R. Bodik. Fast and parallel web page layout. In *WWW*, 2010.

[25] R. Song, H. Liu, J.-R. Wen, and W.-Y. Ma. Learning block importance models for web pages. In *WWW*, 2004.

[26] N. Thiagarajan, G. Aggarwal, A. Nicoara, D. Boneh, and J. Singh. Who killed my battery: Analyzing mobile browser energy consumption. In *WWW*, 2012.

[27] Z. Wang, F. X. Lin, L. Zhong, and M. Chishtie. Why are web browsers slow on smartphones? In *HotMobile*, 2011.

[28] Z. Wang, F. X. Lin, L. Zhong, and M. Chishtie. How far can client-only solutions go for mobile browser speed. In *Proc. WWW*, 2012.

Scout: An Asymmetric Vehicular Network Design over TV Whitespaces

Tan Zhang Sayandeep Sen Suman Banerjee

Department of Computer Sciences
University of Wisconsin-Madison, USA
{tzhang, sdsen, suman}@cs.wisc.edu

ABSTRACT

We explore the use of TV whitespaces based communication systems for providing robust connectivity to vehicles. We envision a setup where whitespaces base stations mounted by roadside that communicate with whitespaces gateway nodes placed on vehicles. A key challenge in this setup is the *asymmetry in transmission power limits* – the fixed base station is allowed to communicate at up to 4W, while the mobile gateways on vehicles are limited to 100mW. This paper presents a specific network design called *Scout* to deal with this asymmetry in which whitespaces transceivers are used in the downlink direction while a more traditional cellular path is used in the uplink one. As the key component of this system, we describe a novel channel probing mechanism that sends a forward radio to look ahead and identify the best channel parameters to be used when the rear radio eventually reaches the forward post. We report various challenges and experience in this design and our ongoing plans to use it for providing Internet access to public city buses. Our initial results indicate a 4× coverage improvement and 1.4× throughput gain achieved by *Scout*.

1. INTRODUCTION

TV whitespaces (512-698MHz), recently released by FCC for unlicensed usage in the U.S. [6], provides wireless communication systems with abundant spectrum resource (180MHz) and excellent propagation characteristics (1.9km). In this work we intend to leverage this whitespaces spectrum for providing long-range and high-speed network connectivity to moving vehicles.

The application we target at is to provide Internet access to the commuters of a city metro with about 250 buses. Figure 1 shows our target deployment, which consists of base stations mounted along roadside, connected to a proxy server. The base stations backhaul Internet traffic generated by the whitespaces gateway nodes (clients) mounted on top of the buses. Each gateway node is connected to a WiFi hotspot inside the bus for providing Internet access to commuters.

Ideally, we would like to use a whitespaces-only solution. However, due to various scalability and cost reasons, we instead adopt an asymmetric network design called *Scout* which uses whitespaces for downlink communications and the cellular path for uplink communications. In particular, we present two key ideas in *Scout*: (i) the use of a cellular link to significantly extend the coverage of each whitespaces base station, (ii) the use of an extra scouting radio to compensate for the effects of feedback delay over the cellular link in taking protocol decisions.

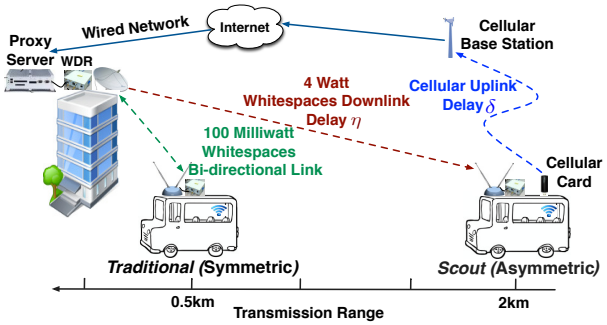


Figure 1: *Traditional* (symmetric) Whitespaces Network v.s. *Scout* (asymmetric) Whitespaces Network. The symmetric network (left vehicle) uses whitespaces for both uplink and downlink communications. In contrast, *Scout* (right vehicle) uses whitespaces for downlink communications, while using the cellular service for uplink communications. The delay of the cellular feedback path $\delta \gg \eta$, the delay of whitespaces downlink.

1.1 Motivation and design of Scout

One major design goal of this vehicular network is to leverage the good propagation characteristics of TV whitespaces frequencies to reduce the deployment and management cost by having a small number of base stations to cover a large area. Unfortunately, the *asymmetric transmission power limit* in TV whitespaces has significantly limited the reachability of the base station. According to the FCC's recent ruling [6], the mobile client is allowed to transmit at *much lower* power–$100mW$ [1] compared to the static base station allowed to–$4W$. The 40× difference in the power limit is to protect against the potential interference resulting from the mobile clients roaming into some unpredictable locations. Thus, a symmetric, whitespaces-only network solution as proposed in [8] (Figure 1 left vehicle) would limit the transmission range of the base stations to that of the "weak" mobile clients when bi-directional communication applications are commonly supported. Measurements in our whitespaces testbed have shown 4× reduction in the transmission range of the base station ($1.9km$ and $0.5km$ for 4W and 100mW

[1]This maximum power includes the gain of antennas and regardless of number of channels used for communications.

ACM HotMobile'13, February 26–27, 2013, Jekyll Island, Georgia, USA.

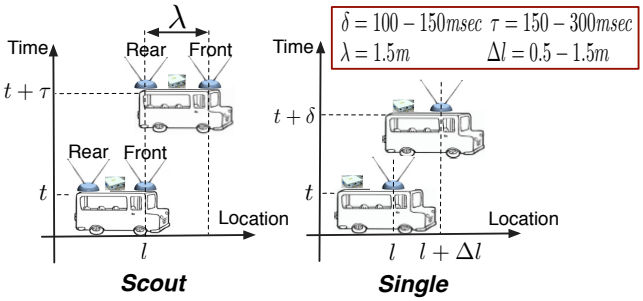
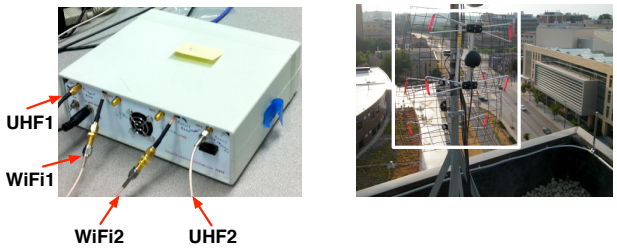


Figure 2: Diagram of *Scout* and *Single*. λ is the antenna separation, τ is the antenna alignment period and δ is the cellular delay. $\tau \geq \delta$.



(a) Wideband digital radio. (b) The base station.

Figure 3: Experiment setup.

respectively) in a symmetric network design. To address this problem, we design *Scout* which has the following characteristics.

(i) An asymmetric network design for improving the coverage of base stations: Figure 1 (right vehicle) shows *Scout*, an asymmetric network architecture to deal with the problem of power asymmetry. *Scout* uses different technologies for uplink and downlink communications. The downlink connection (from the base station to the bus) is over whitespaces frequencies, while the uplink (from the bus to the base station) is over cellular. Thus, *Scout* leverages the ubiquitous cellular connectivity to circumvent the "weak" whitespaces uplink, in turn enabling each whitespaces base station to extend the bi-directional communication range to a maximum. Furthermore, since the downlink load is reported to be dominant in many network applications (10× in WiRover [7]), *Scout* is efficient in utilizing whitespaces spectrum to transmit the majority of data. In our current implementation, we use a WiFi-based whitespaces hardware in [16] for downlink and 3G for uplink communications.

Unfortunately, the operation of the asymmetric network is plagued by the *high feedback latency* in the cellular uplink, which is 10–500× of the packet transmission duration in whitespaces communications. The problem is exacerbated by vehicle mobility which causes rapid change of communication environment. Since most of wireless communication systems rely on channel feedback for making protocol decisions at different layers of the stack, e.g., PHY data rate, FEC at the MAC layer as well as congestion control, error recovery at the transport layer, the poor feedback accuracy can significantly degrade backhaul performance. To tackle the problem of slow uplink, we present our second technique of using an extra radio to measure channel condition for a location in advance. This information is then used for optimizing various transmission decisions for another radio by the time it reaches that specific location.

(ii) A scouting radio for channel estimation: The core intuition of our work comes from the observation that the location of a vehicle has a profound effect on the channel characteristics experienced by a radio mounted on the vehicle [11, 12, 14]. For instance, a vehicle traveling behind a building is observed to have much worse reception than driving in line-of-sight to the base station. To leverage this channel property, we place a scouting radio at the head of the vehicle (front radio) as shown in Figure 2. This front radio passively monitors the downlink traffic at the current location l. It sends back the observed channel information, e.g., packet loss rates, to the base station over the cellular link. After time τ, the receiving radio placed at the rear of the vehicle (rear radio) reaches the same location l. The base station can utilize the channel observation made at l, τ time ago to adjust its transmission for the rear radio.

While the proposed scheme is incapable of tracking fast fading, we will show in Section 2 that channel estimates made by the scouting radio is still beneficial in estimating different link characteristics. This in turn leads to better protocol decisions and ultimately improves the overall performance of the vehicular Internet access. Furthermore, the idea of adding one scouting radio is complementary to various diversity combining techniques used in existing multi-antenna systems, which we will discuss in Section 6.

In this paper, we highlight the efficacy of the scouting radio based mechanism by showing that it can enhance the performance of rate adaptation. We emphasize that *while the problem of rate adaptation has been well studied in a traditional, symmetric communication system, it is unique in our asymmetric network with a feedback link hundred times slower than the forward path.*

Comparison with cellular technologies: Finally, we note that cellular technologies (3G, LTE, WiMax, etc.) also successfully address the problem of providing long-range network coverage to mobile clients with weak transmission power. Unfortunately, translating the gains achieved in *licensed bands* by the cellular technologies to operate over *opportunistic* TV whitespaces appears nontrivial. One of the major challenges is that traditional cellular technologies require centralized coordination by the base station to tightly control the transmission power of each mobile station [5, 18] for managing interference. This allows low power mobile clients to communicate successfully with cellular base stations. However, in the whitespace domain the base station would be incapable of controlling other unlicensed devices potentially using an entirely different communication protocol. The uncoordinated interference is likely to overwhelm the weak uplink signal. *Scout* is robust to interference by communicating the uplink traffic over the licensed spectrum.

In contrast to a symmetric cellular solution over the licensed spectrum, *Scout* offloads the majority of traffic over TV whitespaces. The abundant free spectrum leads to better performance and lower spectrum cost if a pre-paid cellular billing option is available.

1.2 Contributions

Our contribution in this work is two-fold. First, we present an asymmetric network architecture–*Scout* which efficiently uses TV whitespaces for providing wide-area network connectivity to vehicles. Second, as a key component in *Scout*, we propose a novel channel probing framework to address the problem of feedback delay, enabling our system to extract most benefits out of TV whitespaces. Based on our trace driven simulation, *Scout* can extend the coverage of the base station by 4× and achieve a median throughput improvement of 1.4×.

2. FEASIBILITY OF SCOUT

In this section, we validate the feasibility of *Scout*. We begin by explaining our experiment setup.

2.1 Experiment setup

Radio platform: Figure 3(a) shows our radio platform, called the Wide Band Digital Radio (WDR). It performs a frequency translation function similar to the KNOWS platform [1]. With two independent signal processing paths, the WDR can simultaneously translate two signals between the UHF band and the ISM band. This enables us to use a single WDR radio to process signals received from both antennas at the client. The converted signals are fed to WiFi cards for the baseband processing. Due to some hardware limitations, the current version of WDR can support 802.11 b/g data rates up to 18Mbps.

Testbed: Our testbed currently includes a base station and a mobile client. The base station consists of a host machine, a WDR radio, a high-gain power amplifier and a directional TV antenna, as shown in Figure 3(b). The total transmission power at the base station is 3.8W. For the mobile client, we use a vehicle carrying one WDR radio and two omni-directional antennas. The downlink communications are configured at a center frequency of 662MHz with a bandwidth of 20MHz according to a spectrum occupancy database [19].

Metrics: We measure the loss rate calculated for every 10 contiguous packets at the same fixed data rate. We use it as an indicator of channel quality for a given location and at a given time. We then calculate the magnitude of difference in loss rates under different time separations to classify whether channel condition has changed with varying location or time (or both). We denote this time separation as a *lag* in the following discussion.

2.2 Validation of intuition

The feasibility of *Scout* can be ascertained by comparing the following two approaches, which are shown in Figure 2. In *Scout*, suppose the front radio measures the loss property at location l, time t. How accurate is this measurement in predicting the channel condition for the rear radio when it reaches the same location l at time $t + \tau$? We contrast this with the other alternative of a single radio (*Single*). In the single radio scenario, the only radio will estimate the loss property at location l at time t, and use this estimate to predict its performance at location $l + \Delta l$ at time $t + \delta$.

To understand the stability of channel loss properties as only a function of time, we present the variation of loss rates for the same locations with different lags. We measure this by placing a single radio mounted atop a car at 12 equally spaced locations on a 200m road stretch. Figure 4 (*Single static*) shows that loss variation remains small with a lag below 300*msec* for all the measured locations.

We next determine the stability of loss measurements done by the same single radio as a function of both time and location. The speed of the vehicle is between 5m/s (18 km/hr) and 10m/s (35 km/hr), which is typical for urban area due to the 40km/hr speed limit. As can be seen from Figure 4 (*Single* 10m/s and 5m/s), the difference in loss rates increases drastically with increasing lags. The degree of variation is expected as the single radio system is measuring the loss rates at different locations and different time. When using the stale channel observation to predict the loss rate, *Single* would make an estimation error of over 30% under the delay of 100–150*msec* in a 3G uplink.

We finally benchmark the mismatch in the loss rates under *Scout* setup with two radios (front and back) *at the same location* with different lags. The result is again shown in Figure 4 (*Scout variable speed*). We note that the difference in loss rates between two radios at the same location with a lag of 300*msec* remains a fifth of a single radio traveling at 10m/s speed. This demonstrates that *Scout* can indeed improve the channel estimation at a given location.

Based on this maximum lag, we choose the antenna spacing λ to be 1.5m and the driving speed 5–10m/s. This ensures that the highest τ $300msec$ $(1.5\text{m}/5m/s)$ to be within this maximum threshold, and the lowest τ $150msec$ $(1.5\text{m}/10m/s)$ to be greater than the typical cellular delay δ (100*msec*).

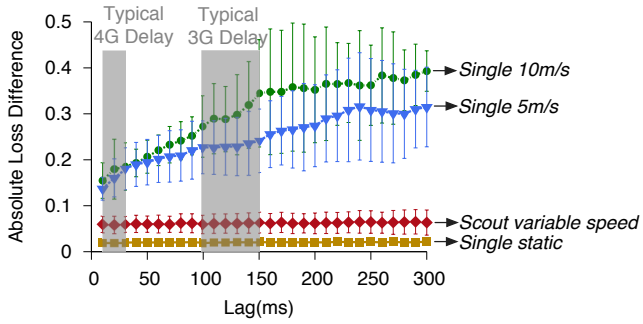


Figure 4: Average of absolute loss difference at various lags for 12Mbps packets. The lag is the elapsed time between two measurements. *Single* denotes one radio and *Scout* is the two radio setup.

Finally, we note that adopting 4G LTE as the feedback link would introduce much lower latency (10–25ms). However, Figure 4 shows that it still leads to 15% - 20% estimation error which *Scout* can avoid. This justifies the effectiveness of *Scout* along the advancement of the cellular technology.

3. DESIGN OF SCOUT

In *Scout* we use a single radio at the base station for downlink transmission. At the client, we use the rear radio for packet reception, while leveraging the front radio to passively monitor the downlink traffic for channel estimation. The packet acknowledgments generated by both radios are sent over the cellular link to the base station.

To select a data rate for the rear radio at a given location l, we use the feedback previously generated by the front radio at l, combined with the feedback from the rear radio currently available at the base station. We use a GPS device at the client to relate a feedback from the front radio to the monitored location. The GPS device reports its reading once every second over the cellular uplink to the base station. We present the pseudo-code of *Scout* in Algorithm 3 and give a detailed explanation next.

Combining feedback from dual radios: In our deployment we observe mismatched reception performance between two radios when their antennas are mounted at different locations on top of the vehicle, e.g., different orientation and tilting. This difference is usually small enough for the two radios to experience a common channel trend. However, we find that selecting data rates solely based on the feedback from the front radio leads to sub-optimum performance.

In *Scout* we utilize the front radio to detect channel variation, while using the rear radio to identify a set of rate candidates to select from. Specifically, we use one of the existing algorithms, e.g., SampleRate, to select a preliminary data rate based on the stale feedback from the rear radio. This is shown in Line 1 of Algorithm 3. This rate is expected to work reasonably well under stable channel condition. Once we detect the change of channel condition at the front radio, we use the detected channel trend and the set of candidate rates successfully received by the rear radio to derive a more appropriate rate decision.

Relating feedback to the observed location: The accuracy of associating the feedback to its monitored location significantly affects

Algorithm 3.1: SCOUT($t, \lambda, v, w, \mathcal{F}, \mathcal{R}$)

INPUT t: Current time, λ: Antenna spacing.
v: Vehicle speed, w: Window size.
$\mathcal{F}$: Set of ACKs from the front antenna.
$\mathcal{R}$: Set of successful rates from the rear antenna.
OUTPUT $rate$: Selected rate.

$\tau = \lambda / v$
$rate \leftarrow set\_origin\_rate(\mathcal{R})$ (1)
$W_{cur} \leftarrow \{f_j : f_j \in \mathcal{F}, |j - (t - \tau)| <= w/2\}$ (2)
$W_{prev} \leftarrow \{f_j : f_j \in \mathcal{F}, |j - (t - \tau - w)| <= w/2\}$ (3)
$var\_detected \leftarrow detect\_variation(W_{cur}, W_{prev})$
if $var\_detected < 0$
  then $rate \leftarrow set\_lower\_successful\_rate(rate_{prev}, \mathcal{R})$ (4)
  else if $var\_detected > 0$
  then $rate \leftarrow set\_higher\_successful\_rate(rate_{prev}, \mathcal{R})$ (5)
return $(rate)$

the performance of *Scout*. For example, Figure 4 shows that a location error of 3m (corresponding to 300ms lag at 10m/s speed) can lead to 40% off in loss estimation. Unfortunately, our low priced GPS modules have a positioning inaccuracy on the order of 10m.

To circumvent this problem, we use the speed reading v reported by the GPS instead of the geo-location reading since it has a much higher accuracy–0.1m/s. We calculate the radio alignment period τ, which is the time elapsed for the rear radio to reach the same location l since the front radio was previously at l (Figure 2). We calculate this period with the formula $\tau = \lambda / v$ where λ is the fixed antenna separation. Note that v remains constant in our calculation considering the short GPS updating interval (1*second*). We then retrieve a window of ACK packets W_{cur} generated by the front radio τ time ago, which is the desired feedback observed at location l (Line 2 in Alg. 3). Since τ is small (< 300*msec*), the positioning inaccuracy of *Scout* is below $3cm$ (0.1*m/s* $\times$ 300*msec*).

Detecting channel variation: To this purpose, we analyze the error performance between the feedback W_{cur} generated by the front radio at location l, and another window of ACKs W_{prev} generated prior to W_{cur} (Line 3 in Alg. 3). We compare the packet loss of each common data rate between these two windows of packets. If the loss rate of a given data rate in W_{cur} increases (decreases) by at least α (β) fold over W_{prev}, we conclude that the error performance of that data rate has changed. We use a voting mechanism to combine results for all the data rates to determine the trend of channel variation. We empirically set α and β at a large value–0.5 to prevent *Scout* from reacting to random channel fluctuation. We configure the time duration of W_{cur} and W_{prev} to be 25ms to collect sufficient, yet relevant feedback.

Adjusting data rates in response to channel variation: Based on the detected trend of channel variation, we select the next higher or lower data rate to the previous rate decision, but only from a set of candidate rates recently succeeding at the rear radio (Line 4-5 of Alg. 3). By doing so, we can choose a rate not only suitable for the current channel condition but also consistent with the reception performance of the rear radio.

4. IMPLEMENTATION

We implement *Scout* at a 3.5 layer on top of the cellular and whitespaces links as shown in Figure 6. To provide a single link abstraction, we leverage a virtual network device in Linux called *TUN*, and have the base station and the client exchange application data through it. We create a user-space program passing packets between *TUN* and one of the underlying network interfaces, i.e., cellular and whitespaces. For downlink communications, the program running at the base station reads the application data from *TUN*, sending them through the whitespaces interface (ath0). At the client, the program delivers downlink packets received by the rear whitespaces radio (ath1), and sends them via *TUN* to the application. This client program also generates ACKs for the packets received by both whitespaces radios (ath0 and ath1), sending them with the GPS readings to the base station over the cellular link.

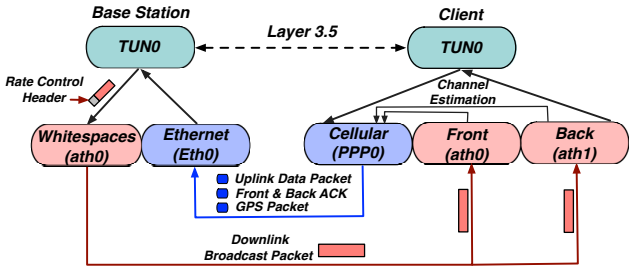


Figure 6: Implementation of *Scout* with *TUN* devices.

We configure the whitespaces radio at the base station to operate in the WiFi broadcast mode. This prevents any client radio from generating MAC layer ACK over the whitespaces, while having both client radios to receive the downlink packets for *Scout*. We enable the base station to control the broadcast data rate by appending a control header in each downlink packet, and modify the Ath5k driver to accept this control information.

We implement the user-space program in 4500 lines of C++ and add about 20 lines of C code in the Ath5k driver.

5. EVALUATION

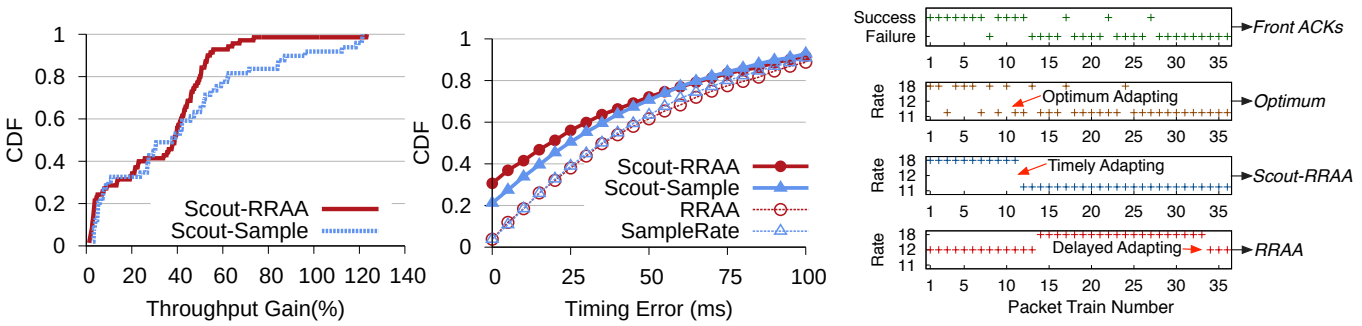
We evaluate the performance of *Scout* after integrating it with two popular rate adaptation algorithms, SampleRate [4] and RRAA [21]. We denote the enhanced algorithms as Scout-Sample and Scout-RRAA. We have found that *Scout* achieves median throughput improvement of 38% and 39% over SampleRate and RRAA.

Methodology: We use trace-driven emulation as a preliminary evaluation of *Scout*, with real trace captured in TV whitespaces. For trace collection, we use a similar approach as in AccuRate [15] by instructing the base station to transmit short back-to-back packets (200 byte), using 8 802.11 b/g data rates up to 18Mbps alternatively. We then use two radios at the client to capture packet traces from 10 drives along a 1.5km bus route. The distance between the base station and the client is about 200–750m, and the vehicle speed is about 18 – 35 km/hr, which is typical for our city metro. We choose the antenna spacing to be $1.5m$ as described in Section 2. We term each set of 8 contiguous packets at all different rates as a *packet train*.

For our emulation, we make each algorithm select one data rate in each packet train. If the chosen rate belongs to one of the successful rates in the current train, we conclude that this rate succeeds in the duration of this train and vice versa. We then calculate the throughput of different algorithms based on these results. To emulate the feedback delay (typically 100ms in our testbed), we provide each algorithm with feedback generated by both radios 100ms ago at the client for rate selection.

We empirically adjust the sampling interval of SampleRate from 10 packets to once every train (8 packets). We further set the size of estimation window for both algorithms to be 10 trains, which is found to perform best in our emulation.

Throughput: We calculate the throughput of different algorithms in each 50m road segment to evaluate performance improvement



(a) Throughput gain. (b) Detection accuracy. (c) Snapshot of rate decisions.

Figure 5: Experiment results. (a) CDF of throughput improvement of Scout-RRAA and Scout-Sample over RRAA and SampleRate. (b) CDF of timing errors in detecting channel variation. (c) A snapshot for rate decisions made by different algorithms.

of *Scout* under different road conditions [2]. The average throughput from 10 drives is 5.5Mbps and 5.2Mbps for Scout-Sample and Scout-RRAA respectively. Figure 5[a] shows the CDF of throughput improvement achieved by Scout-Sample and Scout-RRAA over SampleRate and RRAA. We observe a median throughput gain of 38% and 39%, and an upper quartile gain of 57% and 48%. We expect more improvement when higher data rates are available.

Timeliness in adapting to channel variation: To this purpose, we benchmark the time offset of different algorithms to an optimum algorithm in adapting to channel variation. The optimum algorithm selects the highest successful data rate in each train. Specifically, we record the time of data rate changes made by different algorithms, and measure the absolute time difference for each rate change between algorithms under study to the optimum algorithm. We denote each time difference as a timing error. Figure 5[b] shows the CDF of timing errors of different algorithms. We observe 31% and 21% of the case when Scout-RRAA and Scout-Sample have detected channel variation at the exact time. In contrast, RRAA and SampleRate have only detected 4% of channel variation in a timely fashion. Thus, Scout-RRAA and Scout-Sample have achieved a 8× and 5× improvement in detection accuracy. We further report that the two *Scout* algorithms incur 5% more false positives than their original counterpart for detecting channel variation.

Snapshot of different algorithms in operation: We finally present a snapshot of rate decisions made by different algorithms in Figure 5[c]. First, we can observe a similar trend of channel condition observed by the two radios at the same location, when referring to ACKs previously generated by the front radio (Front ACKs) and the data rates selected by the optimum algorithm for the rear radio (Optimum). Second, Scout-RRAA accurately detects the channel variation, and adapts to a lower rate at the same time as the optimum algorithm does. In contrast, RRAA selects a lower rate at a much later time of 23 trains (about 115ms) due to the feedback delay. Finally, Scout-RRAA directly selects 11Mbps rate based on those candidate rates successfully received at the rear radio, rather than choosing the ineffective sequential rate–12Mbps as RRAA does.

6. DISCUSSION AND FUTURE WORK

Improving channel estimation: In *Scout* the front radio at each mobile client monitors the downlink traffic broadcasted to *all the clients* for channel estimation. We expect this downlink traffic to be highly available when the number of clients served by a base station is high. As a performance enhancement, we intend to develop a mechanism for the base station to periodically send probe traffic upon detecting the downlink to be idle. Furthermore, we plan to investigate more sophisticated indicators, e.g., channel response for detecting channel variation.

Fully leveraging the dual radios: The *Scout* design currently uses one extra radio (front) solely for channel estimation. In our future work we intend to investigate different schemes in combining signals received by both radios to improve reception diversity, e.g., via maximum ratio combining [20] at the PHY layer or packet level combining [9] at the MAC layer.

Scouting at different layers: We intend to generalize the application of *Scout* feedback to enhance the performance of all layers of the network stack. For instance, at the MAC layer we can potentially apply forward error correction codes (FEC) and proactively duplicate downlink packets based on this feedback information. Similarly, at the transport layer we will leverage *Scout* to detect connection "blackout", and predicatively offload the traffic over the cellular link to prevent the slow start of TCP.

7. RELATED WORK

We divide the related work into five categories, i.e., whitespaces networking, vehicular networking, multi-antenna systems, channel estimation and rate adaptation.

Whitespaces networking: A few research efforts [1, 10] have explored designing networks over TV whitespaces. WhiteFi [1], as the first whitespaces network, uses a symmetric network design with a WiFi-like protocol. SenseLess [10] is a network design that purely relies on a spectrum occupancy database to determine available channels. Built on these two pieces of work, *Scout* addresses the challenge of extending the coverage of a vehicular network, constrained by the unbalanced transmit power limit in a given free channel. In contrast to WhiteFi, *Scout* uses an asymmetric architecture consisting of one whitespaces link and one cellular link.

Vehicular networking: A large body of prior work [2, 3, 7, 11, 13] has been done for providing Internet connectivity into vehicles. The existing approaches can be categorized into cellular based solutions (MAR [13], WiRover [7]), WiFi based solutions (ViFi [3], MobiSteer [11]) or a combination of both (Wiffler [2]). In contrast to all these approaches, *Scout* explores a whitespaces based backhauling solution backed up by a cellular uplink. It achieves long-range and high-speed network connectivity by harvesting both the long propagation range (1.9km) and the abundant spectrum resource (180MHz) in TV whitespaces.

Multi-antenna systems: The design of multi-radio, multi-antenna

[2]The reason for choosing the road segment to be 50m is to ensure that at least 5s trace is available for calculating each throughput sample.

systems [9, 17, 20] have been well studied in the past work. These MIMO systems harness the path diversity in wireless channel for robust reception [9, 20], scaling throughput [20], or simultaneous communications with multiple users [17]. *Scout* is different from all these MIMO techniques by using an extra radio system for channel estimation. More significantly, *Scout* is complementary to all these techniques.

Location-based channel estimation: *Scout* leverages a location dependent channel property, which has been reported in the prior work [11, 12, 14]. Bartendr [14] and BreadCrumbs [12] use the location of the mobile client to predict network connectivity in cellular networks and WiFi respectively. MobiSteer [11] uses the location of a vehicle to select the best AP and the directional beam to serve the vehicle. All the above approaches require a training database which is updated on a coarse timescale, e.g., order of days [14]. In contrast, *Scout* can obtain "fresh" channel information collected shortly ago (150–300ms), achieving higher accuracy without any training overhead.

Rate adaptation: We integrate *Scout* with two popular rate adaptation algorithms, i.e., SampleRate [4] and RRAA [21], to demonstrate its efficiency. SampleRate periodically picks a random rate to probe the channel, and selects the rate with the highest throughput. RRAA tracks the packet loss in a short time duration, and uses the predetermined loss threshold to adapt rate. To be effective, both require timely feedback, which is hampered by the cellular delay, yet is benefited by the scouting radio.

8. CONCLUSIONS

In this work we explore an asymmetric network design called *Scout* for providing wide-area vehicular network connectivity over TV whitespaces. The proposed architecture circumvents the bottleneck of the whitespaces uplink with the cellular technology, and significantly extends the transmission range of the base stations. To deal with the feedback delay in the cellular uplink, we have designed a novel channel estimation framework that uses one scouting radio to measure the channel condition at a location beforehand. It provides the base station with this more accurate channel information to select transmission parameters for the receiving radio. Based on our trace-driven simulation, *Scout* leads to 4× improvement in network coverage and 1.4× improvement in downlink throughput.

Acknowledgments

We would like to thank Peter W. Wolniansky for providing us with the radio prototypes operating in TV whitespaces. We are also grateful to our shepherd Matt Welsh whose feedback helped bring the paper to its final form. Tan Zhang, Sayandeep Sen and Suman Banerjee are supported in part by the US National Science Foundation through awards CNS–1040648, CNS–0916955, CNS–0855201, CNS–0747177, CNS–1064944, and CNS–1059306.

9. REFERENCES

[1] P. Bahl, R. Chandra, T. Moscibroda, R. Murty, and M. Welsh. White space networking with wi-fi like connectivity. In *SIGCOMM*, 2009.

[2] A. Balasubramanian, R. Mahajan, and A. Venkataramani. Augmenting mobile 3g using wifi. In *MobiSys*, 2010.

[3] A. Balasubramanian, R. Mahajan, A. Venkataramani, B. N. Levine, and J. Zahorjan. Interactive wifi connectivity for moving vehicles. In *SIGCOMM*, 2008.

[4] J. C. Bicket. Bit-rate selection in wireless networks. In *Master's thesis, Massachusetts Institute of Technology*, 2005.

[5] FCC. Ieee standard for local and metropolitan area networks, 2009.

[6] FCC. Unlicensed operation in the tv broadcast bands, second memorandum opinion and order, 2010.

[7] J. Hare, L. Hartung, and S. Banerjee. Beyond deployments and testbeds: experiences with public usage on vehicular wifi hotspots. In *MobiSys*, 2012.

[8] Microsoft Research. Networking over white spaces. http://research.microsoft.com/en-us/projects/KNOWS/deployment.aspx.

[9] A. Miu, H. Balakrishnan, and C. E. Koksal. Improving loss resilience with multi-radio diversity in wireless networks. In *MobiCom*, 2005.

[10] R. Murty, R. Chandra, T. Moscibroda, and P. V. Bahl. Senseless: A database-driven white spaces network. *IEEE Transactions on Mobile Computing*, 2012.

[11] V. Navda, A. P. Subramanian, K. Dhanasekaran, A. Timm-Giel, and S. Das. Mobisteer: using steerable beam directional antenna for vehicular network access. In *MobiSys*, 2007.

[12] A. J. Nicholson and B. D. Noble. Breadcrumbs: forecasting mobile connectivity. In *MobiCom*, 2008.

[13] P. Rodriguez, I. Pratt, J. Chesterfield, R. Chakravorty, and S. Banjeree. Mar: A commuter router infrastructure for the mobile internet. In *Mobisys*, 2004.

[14] A. Schulman, V. Navda, R. Ramjee, N. Spring, P. Deshpande, C. Grunewald, K. Jain, and V. N. Padmanabhan. Bartendr: a practical approach to energy-aware cellular data scheduling. In *MobiCom*, 2010.

[15] S. Sen, N. Santhapuri, R. R. Choudhury, and S. Nelakuditi. Accurate: constellation based rate estimation in wireless networks. In *NSDI*, 2010.

[16] S. Sen, T. Zhang, M. Buddhikot, S. Banerjee, D. Samardzija, and S. Walker. A dual technology femto cell architecture for robust communication using whitespaces. In *DySpan*, 2012.

[17] C. Shepard, H. Yu, N. Anand, E. Li, T. Marzetta, R. Yang, and L. Zhong. Argos: practical many-antenna base stations. In *Mobicom*, 2012.

[18] C. Smith and D. Collins. *3G Wireless Networks*. McGraw-Hill, 2001.

[19] Spectrum Bridge Inc. Spectrum bridge database. http://www.spectrumbridge.com/Home.aspx.

[20] P. Thornycroft. Designed for speed: Network infrastructure in an 802.11n world. In *Aruba Networks white paper*, 2007.

[21] S. H. Y. Wong, H. Yang, S. Lu, and V. Bharghavan. Robust rate adaptation for 802.11 wireless networks. In *MobiCom*, 2006.

Social Vehicle Navigation: Integrating Shared Driving Experience into Vehicle Navigation

Wenjie Sha, Daehan Kwak, Badri Nath, Liviu Iftode
Department of Computer Science
Rutgers University
Piscataway, NJ 08854-8091
{wenjie, kwakno1, badri, iftode}@cs.rutgers.edu

ABSTRACT
In this paper, we propose a Social Vehicle Navigation system that integrates driver-provided information into a vehicle navigation system in order to calculate personalized routing. Our approach allows drivers registered into certain vehicle social network groups to share driving experiences with other drivers using *voice tweets*. These tweets are automatically aggregated into *tweet digests* for each social group based on location and destination. While listening to the tweet digests, a driver can instruct the social navigator to avoid or choose certain road segments in order to calculate a personalized route. We present our initial design along with a simple prototype implemented for the Android platform.

Categories and Subject Descriptors
H.4.m [Information Systems Applications]: Miscellaneous; K.4.m [Computers and Society]: Miscellaneous

General Terms
Design, Human Factors

Keywords
Social networks, vehicular networks, navigation systems, human-computer interaction.

1. INTRODUCTION
With the ever-expanding affordability of cars throughout the world, traffic congestion is a severe problem which can have a negative impact on the economy, human health and safety, environment, and human productivity. According to a recent study, TomTom published its 2012 1st-quarter congestion index, using six trillion data measurements of real-time data from vehicles. They calculated that Los Angeles drivers spent 33% more commute time when traffic was freely flowing and 77% more time during rush hours [1]. Thus, traffic congestion is a root cause of significant productivity loss, has an adverse effect on an individual's human well-being and energy, and causes vast economic loss.

ACM HotMobile'13, February 26-27, 2013, Jekyll Island, Georgia, USA.

There are a myriad of ongoing attempts to alleviate traffic jams, by using on-ramp flow meters, determining real-time traffic flow using cameras, better radio traffic reporting, improving the road infrastructure, using electronic informational displays along the roadway, etc. In addition, the navigation system manufacturers are integrating traffic-related specialized functions into on-board vehicle navigation systems. In-vehicle navigation systems are predicted to quadruple in North America by 2019, growing to about 13 million [2]. Apart from the traditional navigation manufacturers, such as TomTom or Garmin, drivers are moving to less expensive smartphone navigation apps, such as Google Maps and Apple Maps [3]. Vehicles equipped with these types of devices are expected to be an integral part of the Internet in the near future [4].

Today, most navigation systems and traffic apps can calculate the best route taking into account real-time traffic flow data, as well as historic data to predict traffic flow. For example, Google Maps calculates the current traffic condition using both real-time data from anonymous GPS-enabled device users and historic traffic data to provide optimal routes. Despite this, users may still not have the choice of the route should they have had access to more information, such as the road condition, or the actual reason and status causing the traffic jams. In many cases, such information cannot be determined automatically, while such information can be provided by other drivers driving ahead. For instance, with the information that an accident on a certain road is almost cleared, a driver can choose to stay on that road, even if current traffic is slow, as opposed to another instance where the cause of a traffic jam is a long term lane closure. Just knowing what is happening on the road ahead in a timely fashion can often alleviate stress and significantly improve the driving experience. In the future, driving autonomous or semi-autonomous cars [5,6] will likely make such information exchange among drivers even more desirable and safe at the same time.

In this paper, we propose *social navigation*, which integrates driver-provided information into a vehicle navigation system to calculate personalized routes. Our approach allows drivers to share driving experiences with other drivers using *voice tweets*. These tweets are automatically aggregated into *tweet digests* for each social group based on location and destination. Finally, while listening to the tweet digests, the driver can instruct the social navigator to avoid or choose certain road segments when calculating the route.

The outline of the paper is as follows: Section 2 provides an overview of Vehicular Social Networks. In Section 3, we motivate the reader with a sample scenario of integrating information provided by other drivers. In Section 4, we present our Social Vehicle Navigation design. In Section 5, we describe our implementation for the Android platform and in Section 6 we discuss related work. Finally, Section 7 and 8 concludes with a discussion and plan for future work.

2. BACKGROUND

Ever since the invention of the first automobile, the power of *mobility* has improved the frequency at which people meet to maintain social relations over even greater distances. In the past, the automobile has been an effective tool for socialization. However, today, as *social networks* have become an essential part of our lives, people who have a common interest can easily form virtual social relations, without having to drive to meet one another. Social networks such as Facebook and Twitter have redesigned how people socially connect with their family and friends with no restrictions to frequency and location. Nevertheless, trends to integrate vehicle and social networking are in development and have garnered increasing attention in recent years [4,8].

Traditional social networking services allow people who share interests to form virtual social communities. Mobile Social Networks [8], on the other hand, take into account the physical location and temporal connectivity. In [7], we proposed to integrate vehicular networks with social networking, calling the result *Vehicular Social Networks* (VSN). In VSN, users can opportunistically form periodic virtual communities based on their interest and commuting patterns.

This paper discusses the integration of vehicular social networks into navigation systems taking into account the *shared driving experiences* and *driver preferences*. VSNs allow drivers to form social groups based on their daily commute patterns. Drivers use these ad hoc social groups to post voice tweets whenever they experience unusual road conditions. The collected voice tweets are then converted into digests and are provided to other users in the same VSN group. Based on this shared information, the driver can input a route preference into their navigation system.

3. MODEL

Consider the example of a road layout as shown in Figure 1:

John commutes to work every day. He has the choice of Route 22 or Route 66 towards his destination. John would like to consider safety first when choosing a route, but unfortunately existing navigators do not provide such information. So, John switches to a *social navigator*, which allows him to benefit from the safety recommendations provided by other drivers. John's social navigator joins two VSN groups, one defined for the route 66 and the other one for route 22.

Lucy, driving on route 66 is experiencing busy traffic due to an accident ahead of her and she shares this information by posting a voice tweet (T1). Around the same time, Sam posts a voice tweet (T2) mentioning that the bridge on route 22 is slippery but luckily, there is not much traffic. Other drivers in the VSN group have already posted tweets (T3-T6) about the traffic accident prior to Lucy's tweet. The server realizes that T2-T6 are tweets on the same traffic accident, so it discards the older tweets meanwhile retaining T2, the latest one. The server aggregates T1 and T2 into tweet digests and sends them to every driver in the group. When the tweet digests are played back, John knows about the conditions on both Route 66 and Route 22. John decides to take route 66 even though the traffic is slow. John makes the decision despite route 22 not having much traffic because he prefers a safe, albeit slow traffic. John tells his social navigator "avoid" after listening to the tweet T2, and "choose" after listening to the tweet T1. Based on these preferences, the social vehicle navigation system recalculates the route for John. Had John's navigation system computed the route simply based on real-time traffic data, he would likely have taken route 22.

Such real-time information sharing service can be made possible using vehicular social network (VSN). Users can join VSN groups that are of interest, and can either post or listen to other users' real-time voice tweets about the traffic. Then, based on the user's perception of the traffic situation, the social navigator can avoid or choose certain routes. To demonstrate its feasibility, we present the design and mechanism of *NaviTweet*, a social vehicle navigator that allows drivers to join a VSN group, post or listen to traffic related voice tweets and consequently include the driver's preference into the navigator's route calculation.

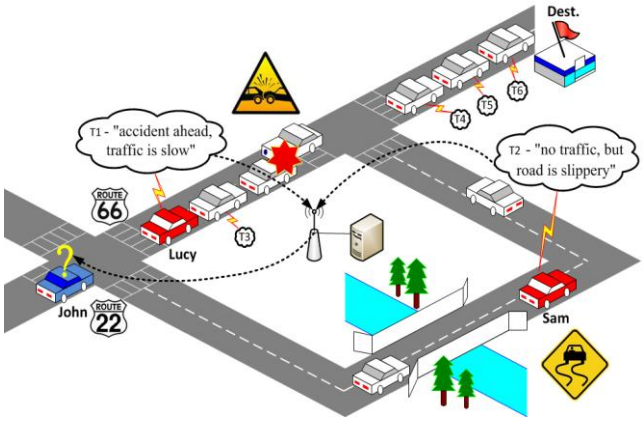


Figure 1 Example Scenario

4. *NaviTweet* SOCIAL NAVIGATOR

A typical workflow of the navigator is depicted in **Figure 2.**

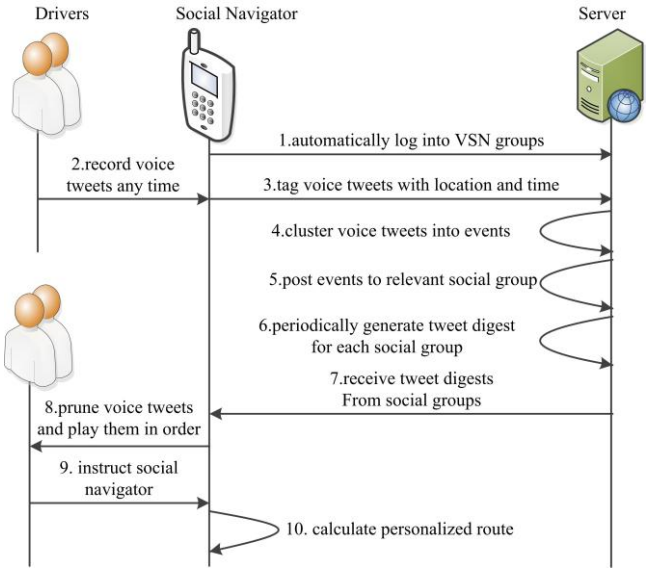


Figure 2. The interaction among the driver, the social navigator and the server

The following steps are performed as part of the workflow:

1. The social navigator automatically logs the driver into the previously registered VSN groups, based on location or destination.
2. The driver records a voice tweet anytime they experience a situation potentially relevant to other drivers.
3. The social navigator tags the voice tweets with the vehicle's location, speed, current time, and driver id, and sends it to the server.
4. The server clusters voice tweets with similar locations and times into events.
5. The server posts events to the relevant social groups.
6. The server periodically generates a tweet digest for each social group out of the most recent voice tweets of each cluster.
7. The navigator receives the tweet digests for the social groups the driver is logged into.
8. The navigator further prunes the voice tweets based on their relevance using criteria such as the vehicle's current location, trajectory and driver's preference for certain group members, then plays them to the driver in the increasing order of distance.
9. After each voice tweet is played, the driver can instruct the social navigator to avoid, choose or ignore it.
10. The social navigator uses this information to automatically calculate a personalized route for the driver.

Each step will be detailed in the following subsections.

4.1 Register and join VSN groups

A vehicular social network (VSN) is a social network of drivers who travel the same set of roadways or have the same destination. NaviTweet uses VSN groups for sharing similar driving experiences. In NaviTweet, we extend the semantics of VSN group definition as follows:

1. Destination group. For example, *group for Manhattan*, or *group for JFK airport*. is characterized by *(i) group name, (ii) center location, (iii) and radius.*
2. Road segment group. Drivers who travel on a major road segment should register the group profiled by the road segment. i.e. *group for NJ turnpike from exit 9 – exit 12 southbound*, or *group for Verrazano-Narrows Bridge eastbound*. Road segment group is characterized by *(i) group name, (ii) road name, (iii) start intersection, (iv) and end intersection*

NaviTweet allows drivers to create VSN groups and manage their memberships through a web portal. We assume that drivers are interested only in the tweets of the VSN groups they have joined. There can be many policies for joining groups. In this work, NaviTweet automatically joins a user to all the VSN groups for which they are registered. Therefore, only those voice tweets will be visible to the NaviTweet client. Further clustering, pruning and sorting on these voice tweets are explained in the following sections. Policies on how to join VSN groups is an interesting issue that we plan to study in the near future.

4.2 Post voice tweet

In our implementation, a voice tweet is limited to 15 seconds. When recorded, a voice tweet is automatically tagged with location, time, speed and driver id. In the current implementation, the driver must touch the screen to begin and end a voice tweet, although a voice-driven interface can also be used.

A voice tweet is posted to the server immediately through a wireless network as soon as the driver finishes recording. Figure 3 shows a typical scenario when many users post voice tweets and download tweet digests every few minutes.

By default, the system expects the tweet to be recorded as close as possible to the event it is referring to in order to allow automatic location tagging. When this is not possible (the driver is busy listening to music, on a call, talking to another passenger, etc.), the location must be explicitly provided. This can be achieved using voice recognition technology or other dedicated user interfaces conducive to drivers.

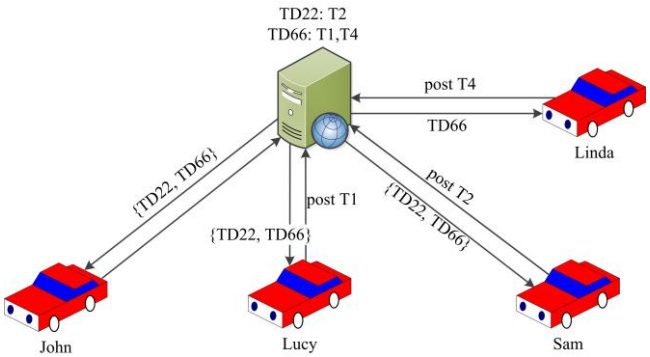


Figure 3. Post voice tweets and receive tweet digests. TD abbreviates for tweet digest. TD22 is the tweet digest on Route 22, and contains voice tweet T2. TD66 contains voice tweets T1 and T4. John has automatically joined groups for Route 22 and Route 66, so he will receive TD22 and TD66.

4.3 Prepare tweet digest

The NaviTweet server periodically prepares a tweet digest for each VSN group. A tweet digest contains selected voice tweets that are recorded near *a destination* or on *a road segment*. For example, in Figure 3, the tweet digest for group 66 only contains selected voice tweets that are published on route 66.

When there is a notable event on the road, multiple users will tend to record voice tweets on that event. This provides a potential opportunity for the server to identify those events by clustering the voice tweets within a short period of time by location. The idea is illustrated in Figure 4. The NaviTweet server clusters all the voice tweets on each road using *DBSCAN* in a two-dimensional space, i.e., distance and time, to identify possible traffic events that triggers the tweeting. In our prototype, we set the minimum tweets per cluster to two to filter out noise and unnoteworthy events. Another important property of traffic events is that they are transient. As the situation evolves, the older voice tweets reflecting past stages of the event will get invalidated. The latest state of the event is likely to be the most relevant one for the drivers. Therefore, NaviTweet packages the voice tweets with the most updated timestamp in each cluster into the corresponding tweet digest for the group. For example, in Figure 4, T1, T4, and T8 are the latest voice tweets for respective events and form tweet digest 66, because they are all on route 66. A NaviTweet client receives the tweet digest but only plays the voice tweets that are on the user's possible routes to the destination. Therefore in

Figure 4, John plays T1, T4 and T8, whereas the other driver only plays T4 and T8. We will further explain the generation of tweet digests on NaviTweet clients in section 4.4. On the other hand, *a noise tweet*, which is a singular tweet that has no neighbor, will never be selected into a digest.

After clustering, selected voice tweets are assigned to the corresponding groups based on their location. NaviTweet server maintains all tweet digests in a hashtable using *groupid* as the key and the set of voice tweets as the value. NaviTweet server updates the tweet digests every five minutes.

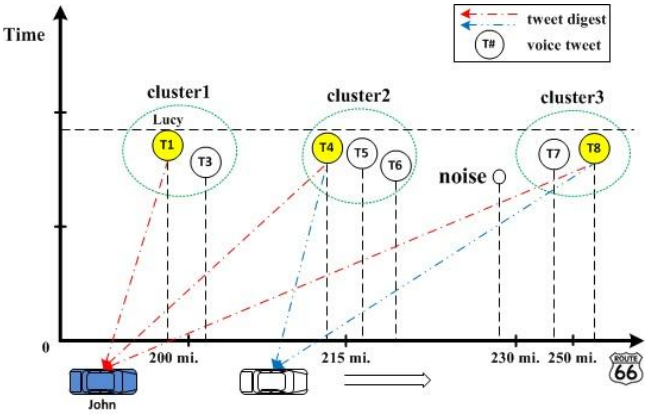


Figure 4. Cluster voice tweets referring to the same traffic event by location and time. T1, T4 and T8 form the tweet digest for the group 66, because they are all located on Route 66. NaviTweet client further prunes the voice tweets in the digests and plays only the relevant voice tweets. The pruning process is explained in section 4.4.

4.4 Play tweet digest

NaviTweet client periodically downloads all tweet digests for the groups the user has automatically joined. The NaviTweet client combines the voice tweets in all digests into a temporary set by excluding duplicate tweets, and matching the set with all possible routes to discard the tweets that do not reside on any of the alternative paths to the destination. After this pruning pass, there will be fewer tweets left in the final *candidate set*.

The calculation of alternative routes is performed using A* search algorithm. Whenever we expand the possible route segment set into an open list, we check if the new segment contains any voice tweets. If so, we attach that segment to the voice tweet meta-data. In the end, NaviTweet client sorts the final *candidate set* in increasing order of a tweet's distance from the current location and selects the top N voice tweets (four in our implementation), which form the local digest that is played**.**

When playing the local digest to the user, it is important to minimize the cognitive load on the driver. Because drivers are more interested in real-time data, we sort the voice tweets in the local digest in the increasing order of distance to the user's current location. We believe it is easier for drivers to process and understand the information in this manner.

Having generated the local digest to be played, NaviTweet client will download the voice tweet audio data from the server and will play each tweet of the digest in that order. The *local digest* is played in the following format.

```
t1: <road name, audio content>;
t2: <road name, audio content>;
                ……
tn: <road name, audio content>;
```

We affix the road name to the voice tweet. In this way, users can figure out which road each voice tweet relates to. We assume users are familiar with the roads they registered for in the VSN.

A new digest is generated and played to the user automatically every few minutes unless she has stopped the navigator. The driver can set the time interval based on how close they want to watch the road events.

4.5 Select and calculate route

As mentioned in Section 3, the criteria for a good route can vary for different drivers on different trips. Therefore, it is crucial to integrate some extent of personalization to the routing engine. NaviTweet client allows a driver to instruct the navigator which road segments to avoid or choose based on what they have heard in the local digest.

After a digest is played, the user will be prompted to compose a decision in the form of "avoid route 1 and 3; choose route 4". Here, route *i* refers to the road segment in the i^{th} voice tweet. When a user says "avoid route *i*", road segment *i* will be penalized by the routing engine, and thus, becomes highly unlikely to be included in the final route. Contrary to this, when a user says "choose route *i*", then the NaviTweet client will select the destination via road segment *i*.

Safety is one of the most important design aspects, so we use speech-to-text recognition for inputting driver's decisions. When a user's command is recognized as irregular input or a user does not provide feedback, NaviTweet client simply ignores the input.

A digest is played and the user decision is considered at the beginning of each interval. There is a slight difference between the first run and later runs. For the first run, after the user's decision is provided, a new route is calculated and rendered; whereas for the later runs, rerouting and re-rendering only applies when the user tries to avoid some segment on the previous route result or chooses some segment that is not contained in the previous result.

4.6 Scalability

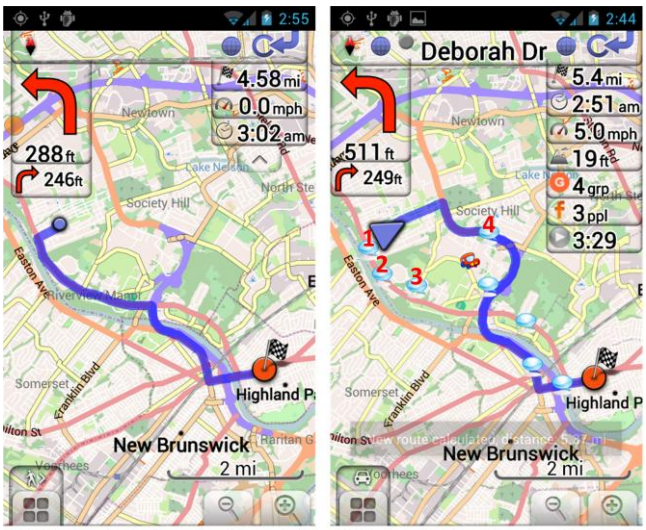
We mention two design issues where scalability comes into play.

1. Tweet Digest. NaviTweet server can easily become bottlenecked if a local digest is generated on the server for each driver. Therefore, we chose to create tweet digest for each VSN group and leave the final selection of the voice tweets to be played to the client. This greatly reduces the number of tweets a client needs to prune and lowers the load of the server as well.
2. Tweet meta-data. The metadata contains a messageid and a timestamp of the voice tweet and is stored in an RDBMS, whereas the voice tweet audio data is stored as a file on a file system. Having generated the final local digest, NaviTweet client retrieves the audio file of each tweet by opening an input stream on the URL locally calculated from the messageid and timestamp of the tweet. Storing the audio data as files organized by timestamp makes archival and clean up much easier, knowing the fact that voice tweets are transient resources.

5. IMPLEMENTATION

We elaborate our work on a freely available open-source GPS navigator, OsmAnd. As a result, our code base only accounts for around 3000 lines of Java code and 2000 lines of PHP, much less than that expected if everything is built from scratch.

NaviTweet server runs on one Amazon EC2 instance using Ubuntu 12.04 LTS. It stores all the metadata in MySQL 5.5, and keeps the voice tweet files on local disk. NaviTweet client runs on Android 2.2+. We use Google Map for VSN group management, and Android speech-to-text recognition and text-to-speech synthesizing library to quickly implement the features that we want. We use OpenStreetMap as our map data to do the pruning and routing. Beyond this, the availability of the source code of OsmAnd and maps was instrumental for implementing the service.



(a) Naïve shortest-path routing from A to B using OsmAnd. The blue dot is the user's current location, and the flag is the destination.

(b) Personalized routing using NaviTweet. The blue bubbles with number are the voice tweets. The red car icon stands for friends' location. User said "avoid route 1, 2, 3; choose 4".
T1: "a dead dear on the road side"
T2: "the traffic is dead"
T3: "a broken tree blocks the road"
T4: "they have Chinese food festival here"

Figure 5. Comparison of traditional routing with NaviTweet

When posting voice tweets and downloading tweet digests to and from the server, there is a network usage cost. Because we limit the length of each tweet message to 15 seconds, an audio message usually requires about 5 to 8 Kbytes using 3GP compression. By default, the digest interval is set at 15 minutes and 4 tweets are played for each digest, so if a user makes 20 voice tweets every day, they will use approximately 6.7 Mbyte every month, which is a reasonable amount of traffic these days.

As a prototype and a first step toward integrating shared driving experiences into the routing engine, NaviTweet provides a smarter way to navigate. Figure 5 demonstrates a real life example of how NaviTweet can help drivers find a better route. We plan to do more sophisticated user studies in the near future.

In the current implementation, to record a voice tweet when a user observes a road event, the user would wave their hand over the screen to trigger the proximity sensor. A microphone image will immediately cover the screen to prompt recording. The user simply records a voice tweet by speaking. The user finishes their recording by touching any part of the navigation interface screen. If the recording exceeds 15 seconds, it ends automatically. NaviTweet takes care of sending the voice tweet to the server automatically in the background. At fifteen minute intervals, a NaviTweet client automatically generates a new tweet digest containing four tweets and plays them in distance-ascending order to the user. Rerouting starts based on the feedback the user gives after hearing the tweet digest.

6. RELATED WORK

The traditional online social networking services such as Facebook, LinkedIn and Twitter focus essentially on providing a foundation of social relations among users who have a common interest without restrictions to where the user is located. Twitter, a combination of an online social networking and microblogging service, allows users to post up to a 140 character text-based message called "tweet" so that the user can join a group to follow a conversation, opinion, story, idea, news or whatever interests the user. Recently, automobiles are integrating social networking services. Developed by Ford, the Twittermobile [9] is a car that can send and receive Twitter messages. Twitter messages sent out by the car can be any type of information ranging from just the driver's mood to informative real-time traffic notices. Also, features such as automatic check-ins via Foursquare or Facebook apps are included. Toyota [10] has also worked on integrating short message social media into the vehicle's dashboard. Both manufacturers are trying to integrate short message social services with cars, yet both are text-based messages. On top of that, the Toyota's version is a predefined template type, for example, "I am going to [destination] and arriving at [time]," where message types have limitations.

In [7], we presented a framework for VSN where people who are physically adjacent to each other construct a periodic virtual social relation. This is an integration of social and vehicular networks whose goal is to virtually build a community for commuters. We built RoadSpeak, a voice chatting system over vehicular social networks, which can be used by daily driving commuters or a group of people who are on a commuter bus or train. NaviTweet, in a similar way, is used to post or listen to traffic related voice tweets, so that the driver's preferences can be incorporated into the navigator's route calculation.

The award winning app for Ford apps competition was presented to students at the University of Michigan. The app is called Caravan Track [11] and has been designed to allow drivers to share vehicle and route information among a group of cars. The idea was derived from the Citizens Band Radio (CB Radio) [12], a short-range radio communications system, where traditionally truck drivers used to locally communicate amongst themselves on topics such as traffic problems, route directions or any other relevant matter. Caravan Track is known also as a tweeting car, allowing members of the group to track one another's specific entities such as location, speed and direction. Route alert functions based on incidents on the road were also applied. The limitation for this work was also the predefined message types where the app had a multiple-choice interface to eliminate typing for safety purposes.

Waze [13] is another popular navigation app that uses crowd-sourcing to provide real-time routing and traffic information along with functions to improve and edit the map itself. Here, social networks are used to send predefined push button messages stating incidents like the degree of traffic, police speed traps or accidents. Chatting functions similar to RoadSpeak [7] are also

available (called ChitChat). Waze incorporates social feed; however, the feed is used in the calculation of the best route and does not accommodate human preference factors to the route selection.

7. DISCUSSION AND FUTURE WORK

There are many issues that require additional research. Building the right user-interface to enable drivers to interact with the social navigator, while driving, is a non-trivial task. Issues such as driver behavior, safety and cognitive load have to be further explored through a systematic user study.

Like any other social feeds, for the system to work, a suitable amount of users are necessary. Incentives for tweeters such as "likes" or points are used in existing apps such as Waze [13] so that many users contribute tweets for the system to work. Such mechanisms can also be integrated in our implementation to properly incentivize tweets. For instance, acquired points (or reputation) of a driver can be used to give priority to their voice tweets in the selection for tweet digests.

Selecting the most relevant tweets to be included in the tweet digest can be particularly difficult when the number of tweets is large. We plan to explore various approaches to achieve tweet selection either automatically, using additional criteria such as user reputation, or semi-automatically, by crowdsourcing this task to people willing to help in real time. Drivers' feedback can also help to eliminate improper or malicious tweets. For instance, when tweets are in large number, multiple tweet digests can be initially created for the same group and distributed to different drivers in order to collect feedback to select the quality tweets.

Finally, security, privacy, malicious users and last but not least passenger safety [14] must also be considered. As future work, we plan to evaluate our NaviTweet on roadways and extend the social voice tweets to link sensor networks (detect environmental pollution) with social networks.

8. CONCLUSION

This paper introduced the social vehicular navigation system that uses driver-provided traffic related voice tweets, an improvement over the current navigation systems that do not have such a feature. Many of the newer navigators do apply real-time traffic data for dynamic route calculation. However, only-computer-based route calculation may not be satisfactory in all situations. Thus, NaviTweet collects voice tweets from those who are in the same vehicular social network groups to allow drivers to share driving experience and decide routes based on personal preference, and suggest routes to the navigation system. We presented the design of NaviTweet, where voice feeds are collected and tweet digests are sent to users in the social group. The driver then instructs the social navigation system to avoid or choose certain routes when calculating a personalized route.

9. ACKNOWLEDGEMENT

We thank our shepherd, Marc Langheinrich and the anonymous reviewers for their insightful comments. This work has been supported in part by the NSF grants CNS-1111811 and CNS-1117711.

10. REFERENCES

[1] Study finds Los Angeles most congested city in North America. *TomTom Press Release*, July 10, 2012, Retrieved Sept. 15, 2012, http://www.tomtom.com/en_gb/licensing/press-room/press-releases/2012/Study-finds-Los-Angeles-most-congested-city-in-North-America.jsp.

[2] Eisenstein, P.A. In-vehicle navigation sales will quadruple by 2019. *The Detroit Bureau*, Sept. 30, 2012, Retrieved Sept. 30, 2012, http://bottomline.nbcnews.com/_news/2012/07/11/12683063-in-vehicle-navigation-sales-will-quadruple-by-2019?lite.

[3] Eaton, K. Map Apps to Bypass the Rivalry of Google and Apple. *The New York Times*, Oct. 3, 2012, Retrieved Sept. 15, 2012, http://www.nytimes.com/2012/10/04/technology/personaltech/navigon-and-waze-map-apps-to-bypass-apple-and-google-review.html?_r=0.

[4] Gerla, M. and Kleinrock, L. Vehicular Networks and the Future of the Mobile Internet. *Computer Networks*, 55 (2), (Feb. 2011), 457–469. DOI=http://dx.doi.org/10.1016/j.comnet.2010.10.015.

[5] Self-Driving Car Test. http://www.google.com/about/jobs/lifeatgoogle/self-driving-car-test-steve-mahan.html.

[6] Coelingh, E., Solyom, S., All Aboard the Robotic Road Train. *IEEE Spectrum*, (November 2012). Retrieved Jan. 3, 2013. http://spectrum.ieee.org/green-tech/advanced-cars/all-aboard-the-robotic-road-train.

[7] Stephen, S., Han, L., Shankar, P. and Iftode, L. RoadSpeak: Enabling Voice Chat on Roadways using Vehicular Social Networks. In *1st Workshop on Social Network Systems*, (Glasgow, Scotland, April 01-04, 2008). SocialNets'08. 43–48. DOI=http://doi.acm.org/10.1145/1435497.1435505.

[8] Vastardis, N. and Yang, K. Mobile Social Networks: Architectures, Social Properties and Key Research Challenges. *IEEE Communications Surveys and Tutorials, PP* (99), (June 2012), 1–17. DOI=http://dx.doi.org/10.1109/SURV.2012.060912.00108.

[9] Quain, J.R. Social Networking for Cars. *The New York Times*, July 20, 2010, Retrieved Sept. 15, 2012, http://wheels.blogs.nytimes.com/2010/07/20/social-networking-for-cars/.

[10] Dowling, J. Twitter comes to cars. *Drive*, Sept. 28, 2012, Retrieved Sept. 30, 2012, http://news.drive.com.au/drive/motor-news/twitter-comes-to-cars-20120928-26p5g.html.

[11] Squatriglia, C. Ford's Tweeting Car Embarks on 'American Journey 2.0'. *Wired*, May 05, 2010, Retrieved Sept. 15, 2012 http://www.wired.com/autopia/2010/05/ford-american-journey/.

[12] CB Radio. http://en.wikipedia.org/wiki/Citizens_Band_radio.

[13] Waze. http://www.waze.com.

[14] Lindqvist, J., Hong, J., Undistracted Driving: A Mobile Phone that Doesn't Distract. In *12th Workshop on Mobile Computing Systems and Applications*, (Phoenix, Arizona, USA March 1-2, 2011). HotMobile'11. 70–75. DOI=http://dx.doi.org/10.1145/2184489.2184504.

Quantifying the Potential of Ride-Sharing using Call Description Records

Blerim Cici[†⋆], Athina Markopoulou[†], Enrique Frías-Martínez[⋆], Nikolaos Laoutaris[⋆]
UC Irvine[†], Telefonica Research(Spain)[⋆]
{bcici, athina}@uci.edu, {efm, nikos}@tid.es

ABSTRACT

Ride-sharing on the daily home-work-home commute can help individuals save on gasoline and other car-related costs, while at the same time reducing traffic and pollution in the city. Work in this area has typically focused on technology, usability, security, and legal issues. However, the success of any ride-sharing technology relies on the implicit assumption that human mobility patterns and city layouts exhibit enough route overlap to allow for ride-sharing on the first place. In this paper we validate this assumption using mobility data extracted from city-wide Call Description Records (CDRs) from the city of Madrid. We derive an upper bound on the effectiveness of ride-sharing by making the simplifying assumption that any commuter can share a ride with any other as long as their routes overlap. We show that simple ride-sharing among people having neighboring home and work locations can reduce the number of cars in the city at the expense of a relatively short detour to pick up/drop off passengers; e.g., for a 0.6 km detour, there is a 52% reduction in the number of cars. Smartphone technology enables additional passengers to be picked up along the way, which can further reduce the number of cars, as much as 67%.

1. INTRODUCTION

Ride-sharing is an effective way to reduce the number of cars on the streets in order to address both individual and city-wide issues. On one hand, individuals are interested in reducing the cost of their car usage and save on gasoline and other usage-based costs [2]. On the other hand, cities are interested in reducing traffic and pollution and provide incentives (e.g. reserved carpooling lanes) to encourage commuters to share rides. In recent years, a plethora of web- or smartphone-based solutions have emerged in order to facilitate intelligent traffic management [18], [17], [5], and in particular ride-sharing. Systems like `carpooling.com`, and `eRideShare.com` have attracted a few million users in Europe and the US but the technology hasn't been widely adopted yet.

ACM HotMobile'13, February 26–27, 2013, Jekyll Island, Georgia, USA.

Ride-sharing systems started in the US during WW-II. These early "word-of-mouth" systems required predefined rendezvous and prior familiarity among commuters, which limited the number of neighbors a person could share a ride with. More recently, web-based solutions, such as `Amovens.com`, allow drivers and passengers to advertise their interest in ride-sharing, thereby increasing the chances of finding a match, but still generally require predefined rendezvous. Using smartphones with ride-sharing apps, such as `Avego.com`, allows drivers and passengers to be matched opportunistically without pre-arranged rendezvous. Such systems are promising but it is still unclear whether they will be widely adopted.

Work in the area has focused on technological, usability, security and legal aspects [16] [20]. Previous research has shown that ride-sharing has economic advantage over driving alone, and that is more spatially flexible and less time consuming than public transportation, but it is not sure if this advantages are strong enough to entice commuters to switch to ride-sharing; privacy and flexibility are major reasons why the vast majority of commuters choose to drive alone. Many believe that current technology provides insufficient levels of security and monitoring to allow people to travel safely with strangers; others believe that it is only an unsolved bootstrapping problem that keeps the technology from booming. Most people, however, implicitly assume that human mobility patterns and the layouts of today's cities exhibit enough route overlap for ride-sharing to take off, once the aforementioned issues are solved.

In this paper, we validate this underlying assumption, which is crucial for the success of any ride-sharing system. To this end, we use home/work locations, for a large population of a city, extracted from a CDR dataset; the inferred home and work locations are used to match people in groups that share the same car. The exact potential of ride-sharing depends on many factors, not all of which can be known at the scale of our study (e.g., behavioral traits). Our approach is to focus only on quantifiable factors and mask all other unknown factors by making the simplifying assumption that ride-sharing is constrained only by the distance of end-points and time. Therefore, our quantification is actually an upper bound of the exact potential of ride- sharing that may be constrained by additional factors.

Our contributions are as follows. We consider two scenarios for ride sharing, and for each scenario we develop an efficient algorithm to do the matching, and quantify the benefit of ride-sharing in terms of reduction in the number of cars. Note that, the theoretical limit of car reduction is 75%

(all users are matched in cars of 4 and only 25% of the cars are used). The scenarios and results are summarized below.

`EndPoints RS`: In this scenario, we consider ride-sharing only among users with nearby (i.e., within distance δ) home and work locations. We formulate the problem of matching users so as to minimize the number of cars, and serve all rides among neighbors that are within distance δ in both home and work location, as a *Capacitated Facility Location Problem with Unsplittable Demand*. Since the latter is an NP-hard problem, we develop an efficient heuristic algorithm that we name `EndPoints RS`. When applied to our CDR dataset, the algorithm provides an upper bound to the ride-sharing potential benefits: 52% car reduction for δ to 0.6 km, if we assume that drivers wait for passengers as long as necessary. Although unrealistic, this puts an upper bound on the savings of ride-sharing based only on spatial information about home and work. When we introduce time constraints into the picture, we find that the effectiveness of ride-sharing becomes proportional to the driver/passenger waiting time for a pick-up, and inversely proportional to the standard deviation of departure times. For example, with a standard deviation of 10 minutes, a wait time up to 10 minutes, and a δ of 0.4 km there is a 26% reduction of cars.

`EnRoute RS`: Next, we compute the routes from home to work using Google Maps and allow en-route pick-ups. First, we match neighbors using the `EndPoints RS` algorithm. Then additional passengers are picked up along the way, which clearly increases the effectiveness of ride sharing, using an iterative algorithm, which we refer to `EnRoute RS`. The same δ and τ applied to end points are applied to en-route pick-ups too. For example, with a δ of 0.6 km the effectiveness on savings jumps from 52% for `EndPoints RS` to 67% for `EnRoute RS`. Taking also into account the randomness in departure times, the corresponding numbers are 35% for `EndPoints RS` and 56% for `EnRoute RS`.

The rest of this paper is organized as follows. In Section II we briefly describe the CDR dataset that is the basis of this study. In Section III, we present our methodology for inferring the Home Work location for individual users. In Section IV and Section V we consider matching based on end points (`EndPoints RS`) and en route (`EnRoute RS`) respectively. In particular, we present efficient matching algorithms and we evaluate the benefits of ride-sharing when applying these algorithms on the CDR dataset. Section VI discusses related work. Section VII concludes the paper.

2. OUR CDR DATASET

A valuable asset of this study is the Call Description Records or CDR Dataset, which we obtained from a large cellphone provider. It contains 820M calls from 4.7M mobile users, during a 3-month period, in the metropolitan area of Madrid, Spain.

CDRs are generated when a mobile phone makes or receives a phone call or uses a service (e.g., SMS, MMS, etc.). Information regarding the time/date and the location of the nearest cell tower are then recorded. More specifically, the main fields of each CDR entries are the following: (1) the originating number (2) the destination number (3) the time and date of the call (4) the duration of the call (5) the latitude and longitude of the cell tower used by one, or both, phones numbers — cell phone companies save CDR records only for their customers. These records are logged for pricing purposes, so they come at no extra cost to the cellular

(a) Headquarters of Telefonica in Madrid

(b) Residential Area: Latitude:$40^o30'13.45''$, Longitude:$3^o38'07.69''$

Figure 1: Example of strictly residential and strictly working areas

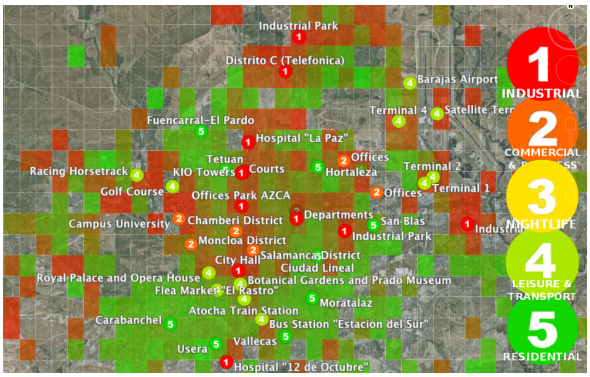


Figure 2: Characterizing Madrid based on our results

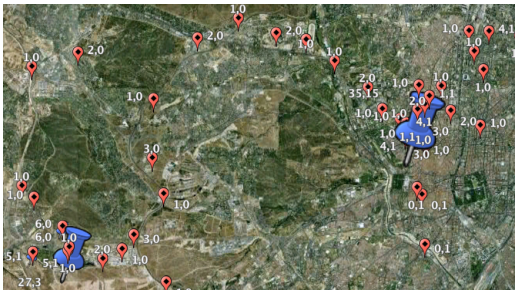
operator. Note that no information about the exact position of a user is known, since cell phone data provide coarse location accuracy — a few hundred meters for city center, and up to 3 km in rural areas. For privacy reasons, no contract or demographic data were made available to us, and the originating and destination phone numbers were anonymized. More details about CDRs can be found in [15].

3. INFERRING HOME AND WORK

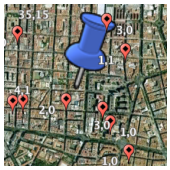
We use an existing methodology to infer the Home/Work locations of 270K individuals in our CDR dataset. This subset of individuals is then used as input to the matching algorithms, discussed in the following section, and the benefit of ride-sharing (reduction in the number of cars) is computed.

3.1 Methodology

We apply the methodology of Isaacman et al. [10], which identifies important places (*i.e.*, places that the user visits frequently and spends a lot of time) in the life of mobile phone subscribers, based on (i) CDR data and (ii) ground truth for a subset of subscribers. More specifically, in [10], the recorded cell towers of a user are clustered to produce the list of places that she visits. Then, regression analysis is applied on the ground truth users (their identified clusters and their true important locations) to determine the features that distinguish clusters that represent important places. Such features include: (1) the number of days that the user appeared on the cluster (2) the duration of user appearances on the cluster (3) the rank of the cluster based on number of days appeared. Once important locations have



(a) A ground truth user



(b) Zooming at home

(c) Zooming at work

Figure 3: The red marks show the recorded cell towers for the user, while the blue marks the clusters. The white numbers next to each mark indicate the number of weekdays and the number of weekends the user appeared in that location. Also, the size of each mark is proportional to the number of days the user has appeared in that location.

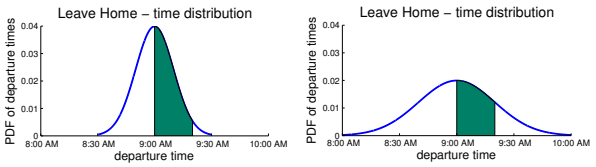
been identified, the authors choose which of these locations is home and which is work. The best features to make this decision are: (4) number of phone calls between 7PM - 7AM (5) number of phone calls between 1PM - 5PM, referred to as *Home Hour Events* and *Work Hour Events* respectively.

In this paper, first, we filter out users for whom we simply do not have enough data: *i.e.* users with less than 1 call per day on average, or less than 2 clusters with 3 days of appearance and 2 weeks of duration. Then, we tune the methodology of [10] in our case. More specifically, we build two classifiers, one for home and one for work, and we train them using the 5 features described above and the ground truth described in 3.2. Once the training is complete, we apply the classifiers to the rest of the users. This was necessary since our ground truth contains only home and work location, while the ground truth of [10] contained other important locations too. Finally, after classification, we keep only the users who have only one location identified as home, and a different location identified as work, since we are interested only in commuters.

3.2 Obtaining the Ground Truth

In [10], a small set of 37 volunteers who reported their most important locations, including home and work. This information was used to train the classifiers that were applied to the remaining dataset of around 170K mobile phone users.

In our case, we did not have access to the actual phone numbers. We obtained our ground truth for a select subset of users based on our knowledge of the city of Madrid. In particular, due to its development pattern in the last 20 years, Madrid has many areas lying around its outer ring highways that are strictly residential and other ones that are strictly industrial. An example of the former are large



(a) When $\sigma = 10$ (b) When $\sigma = 20$

Figure 4: Assuming users u leaves home at 9:10, the users departing with 10 min difference are in the green area under the curve.

residential development projects in previously isolated areas like the one depicted in Fig. 1. Such areas offer a clear distinction between home and work and can be exploited to build our "ground truth". To this end, we selected 160 users that appeared for many days in only one such residential area during 7PM - 7AM (assumed to be "home" hours), and only one such industrial area during 1PM - 5PM (assumed to be "work" hours). Then, the location inside the residential area is pointed as the user's Home, while the location inside the industrial area is pointed as the user's work.

For each one of the 160 users, we visually inspected their recorded locations through Google Earth for a sanity check. In Fig. 3 we show one of these users: this person lives in the location shown in Fig. 3(b), which is the top right blue marker of Fig. 3(a), and work on the location shown in Fig. 3(c), which is the bottom left marker of Fig. 3(a).

3.3 Results

Applying the above methodology to our CDR dataset, we are able to infer the home and work locations of 270K individual users.

The following comparison provides a sanity check. In Fig. 2 we break the city of Madrid into a grid and paint each square of the grid with a combination of green and red. If the number of inferred home locations is higher than the number of work locations, then the color of the square is closer to green, otherwise it is closer to red. We use an existing study [15] to obtain a characterization of locations in Madrid (industrial, commercial, nightlife, leisure and transport, residential). We annotate such areas in Fig.2 using numbered circles, *e.g.*, the headquarters of Telefonica in Madrid is one of the two red circles on the top of the figure. We observe that the squares that we painted red contain more circles indicating industrial and commercial zones, than residential zones. Also, squares painted green contain more residential zones than industrial zones.

However, due to privacy reason, validating home/work remains challenging. We are currently in the process of obtaining permission to compare our results with the billing database of the operator and compute our detection success ratio at the level of postal code.

4. RIDE WITH THY NEIGHBOR

Here we formulate the problem of `EndPoints RS` *i.e.*, ridesharing among people that live and work nearby. We develop a practical algorithm, apply it to the 270K users with estimated home/work locations, and compute the number of cars that can be reduced under different scenarios.

4.1 Formulation

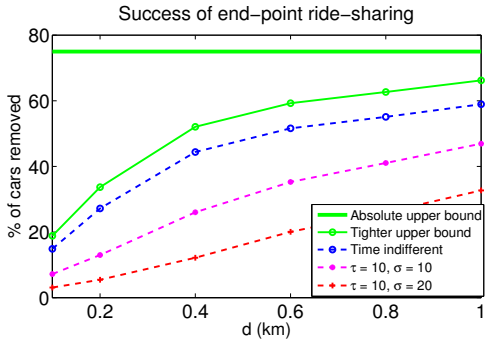


Figure 5: Benefits from `EndPoints RS`.

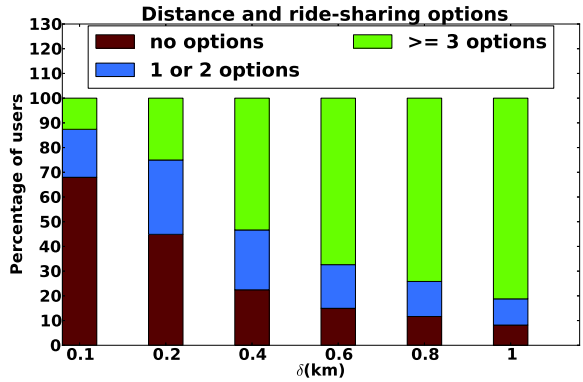


Figure 6: How δ affects the ride-sharing options

Let V denote a set of potential drivers and $c(v)$ the capacity, in terms of available seats, of the car of driver $v \in V$ and $p(v)$ a penalty paid if driver v is selected for driving his car and picking up passengers. Let $h(v,u)$ denote the geographic distance between the home locations of drivers v and u and $w(v,u)$ the corresponding distance between their work locations. Let δ denote the maximum distance between a driver's home/work and the home/work of passengers that he can pick up in his car, *i.e.*, v can have u as passenger only if: $max(h(u,v), w(u,v)) \leq \delta$

Let $d(v,u)$ denote a virtual distance between v and u defined as follows:

$$d(v,u) = \begin{cases} h(v,u) + w(v,u), \\ \text{if} \quad \max(h(v,u), w(v,u)) \leq \delta \\ \\ \infty, \quad \text{otherwise} \end{cases}$$

Our objective is to select a subset of drivers $S \subseteq V$, and find an assignment $a : V \to S$, that minimizes $P(S) + D(S)$, the sum of penalty and distance costs, while satisfying the capacity constraints of cars. The two costs are defined as follows:

$$P(S) = \sum_{v \in S} p(v) \quad \text{and} \quad D(S) = \sum_{v \in V} d(a(v), v)$$

where $a(v) \in S$ is the driver in S that is assigned to pick up passenger v (can be himself if v is selected as a driver). By setting $p(v) > 2\delta \cdot c(v)$ we can guarantee that an optimal solution will never increase the number of cars used in order to decrease the (pickup) distance cost between a driver and its passengers. The above problem is an NP-hard *Capacitated Facility Location Problem with Unsplittable Demand* in metric distance: the set of potential drivers corresponds to the set of locations; the set of chosen drivers corresponds to opened facilities; car capacity corresponds to facility capacity; distance $d(v,u)$ corresponds to the cost of assigning a location v to the facility u. Efficient approximation algorithms are known for this type of facility location problem [14].

The above formulation finds the minimum number of cars needed when there are no timing constraints around departure and return times from home and work. Next we refine the formulation to include time. We assume that departures from home and work follow Gaussian distributions, centered at 9 am and 5 pm respectively, with standard deviation σ. Also, we introduce the wait tolerance τ that captures the maximum amount of time that an individual can deviate from his normal schedule in order to share a ride, Fig. 4. More specifically, if $LH(u)$ expresses the time a person u leaves home to go to work, and $LW(u)$ expresses the time she leaves work in order to return to home. Then, two people u and v, can share a ride only if:

$$max(|LH(u) - LH(v)|, |LW(u) - LW(v)|) \leq \tau$$

The introduction of the temporal constrains will only change the virtual distance between v and u :

$$d(v,u) = \begin{cases} h(v,u) + w(v,u), \\ \text{if} \quad \max(h(v,u), w(v,u)) \leq \delta \\ \text{AND} \quad |LH(u) - LH(v)| \leq \tau \\ \text{AND} \quad |LW(u) - LW(v)| \leq \tau \\ \\ \infty, \quad \text{otherwise} \end{cases}$$

4.2 A practical algorithm

In this section we show how to modify the existing approximation algorithm [14] for the facility location problem described above and obtain a faster heuristic that can cope with the size of our data set.

The existing algorithm starts with an initial random solution and improves it iteratively via local search. At each iteration there are $O(n^2)$ candidate solutions, where n corresponds to the number of potential drivers, and for each one of them we find the assignment (passengers to drivers) that will minimize the cost; this can be done in polynomial time by solving an appropriately defined instance of the *transportation problem*. The algorithm terminates when local search cannot find a better solution.

We modify the algorithm in three ways. First, since the quality of the solution depends mostly on the number of drivers, we try to keep that number as low as possible. Therefore, we use the b-matching [3] algorithm to generate the initial solution, instead of generating it randomly. The input to the b-matching algorithm consists of the set of potential drivers V, a function $p(v)$ that defines the set options for a potential driver v *i.e.* $p(v) = \{u | d(u,v) < \inf\}$, and a global ordering of the potential drivers, O. The global ordering will be based on the number of options; the fewer the options, the higher the position in O. By using b-matching with a global order we are guaranteed to find a solution in $O(n)$ time [3]. For each match generated by b-matching, we assign the potential driver with the most occupied seats to drive; we make sure that every user in V appears in only one car. This solution proves much better than the random

one by paying $O(nlog(n))$ for sorting the users to generate the global preference list and $O(n)$ for the matching.

Second, solving a transportation problem with 270K users is hard. Therefore, we need to modify the local search steps of the approximate algorithm. Given an initial solution we leave the users commuting in cars of four as they are and search for better assignments only for the rest. This way the size of the transportation problem will be reduced and that would speed up the process of generating the assignment.

Third, reducing the size of the transportation problem is not enough; we also need to reduce the neighborhood of candidate solutions. Given an initial set of drivers, S, we create a fixed size neighborhood, where each solution S' is created by doing random changes in S. The reason why we do that is because considering all potential solutions that differ from S only by one, means that we have to examine $O(n^2)$ candidate solutions; that makes each iteration very expensive. Therefore, the fixed size solution helps us speed up the time we spend in each improvement step.

Without the above modifications it would be impossible to solve the problem in real time. Solving an instance of the transportation problem for 270K users required a couple of hours for $\delta = 0.6$ km, and even more when $\delta = 0.8$ or $\delta = 1.0$ km. Therefore, solving $O(n^2)$ such problems for a single iteration becomes too expensive. Moreover, most of the time the solution of the b-matching algorithm was so good that the gain from the improvement steps would be insignificant.

4.3 Results

A this point we are ready to calculate the effectiveness of `EndPoints RS` in the Madrid metropolitan area. We reduce the size of our dataset by randomly selecting only 60% of the users.We do that to capture the fact that only 60% of the population has a car in the area of Madrid [1]. We also show results for the case that half of the car owners use their car at their daily commute (the results are quantitatively close). For the remaining of the section, we will refer to users who can share rides with a specific user v, as *options of* v. Subsequently we compute the percentage-wise reduction of cars

$$\text{success} = \frac{\#(\text{init. cars}) - \#(\text{ride-sharing cars})}{\#(\text{init. cars})} \cdot 100$$

using the following algorithms:

Absolute upper bound: Given our definition of success, we cannot do better than 75%. This is the case when all cars carry 4 people.

Tighter upper bound: All users with at least one ride-sharing option, are assumed to commute in cars of 4.

Time-indifferent matching ($\tau = \infty$): This is the practical algorithm described in Sect. 4.2

Time-aware matching: This is the version of the algorithm that considers timing constraints under the assumption of normally distributed departure times.

In Fig. 5 we see what happens when the drivers are willing to tolerate a detour of δ and deviate τ minutes from their departure times, in order to share the same car with another individual. The results show that with even modest delay tolerance of 10 minutes and detour distance of 0.6 Km (a couple of city blocks) 40% of the cars can be saved. This is more than half of the absolutely optimal performance. Increasing either of the two parameters improves the success ratio. The diminishing improvement with increasing δ can be explained by the number of options users have given the distance δ. In Fig. 6 the red color represents the users with no options, the blue color the users with 1 or 2 options, and the green color the users with 3 or more options. We can see that the success of ride-sharing is proportional to the number of users who have 3 or more options; since, as we can see from Fig . 5 and Fig . 6, they increase in a similar way.

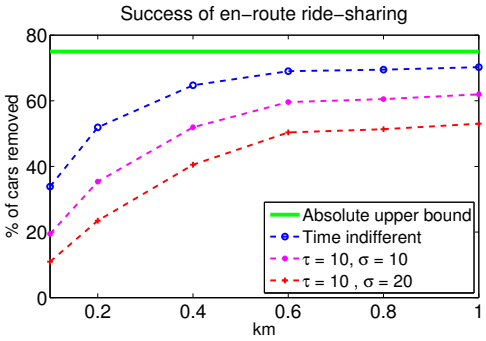


Figure 7: Benefits from `EnRoute RS`.

5. EN-ROUTE RIDE-SHARING

The effectiveness of ride-sharing can be greatly enhanced by picking up additional passengers en-route. For example a driver that lives in a sparsely populated area might not have any neighbors to fill his seats but once he enters the city he might be able to pick several passengers that have routes "covered" by his own. To quantify the benefits of en-route ride-sharing we obtain routes from Google Maps for our dataset of 270K users and extend the algorithm of Section 4.2.

5.1 En-route algorithm

We use an iterative algorithm with the following steps in each iteration.

1. Run the basic `EndPoints RS` algorithm.
2. Exclude from the solution cars that get fully packed (a car of 4). Then order cars in decreasing order of passengers and start "routing" them across Madrid using data from Google maps.
3. When the currently routed car v meets a yet un-routed car v', then v is allowed to steal passengers from v' as long as it has more passengers than v' (a rich-get-richer strategy). Whenever a routed car gets fully packed it is removed from further consideration. Whenever a car with a single passenger is encountered the number of cars is reduced by one.

This is repeated until there is no possible improvement. It can be shown (omitted for lack of space) that the rich-get-richer rule leads to convergence.

5.2 Results

Fig. 7 shows the performance of `EnRoute RS`. To make the comparison with `EndPoints RS` easier we summarize results from both in Table 1: one can verify the significant improvement obtained through `EnRoute RS`, which is several cases comes within 10% of the optimal performance.

6. RELATED WORK

Mining mobility patterns from wireless traces has recently received a lot of attention. Of particular interest to this work is the mining of mobility patterns from CDRs. The best examples of this area are the work of Gonzalez et al. [8], who use CDRs in order to characterize the distribution of human trajectories, and the work of Isaacman et. al. [10] [11] [12] [13] who use CDRs to characterize various aspects of human mobility, such as important places, ranges of travel, etc. The previous examples use only the location information, but recent work [4] [6] also exploits the social graph inside the CDRs also (who calls who). Finally, another example of human mobility form CDR data is [7].

To the best of our knowledge this is the first work on ride-sharing applications based on CDRs. Other ride-sharing applications, that exploit wireless traces, use mostly GPS [9] [19]. The work presented in [9] presents a frequency-based route mining algorithm designed for GPS data and is used to extract frequent routes and recommend ride-sharing plans using the GPS traces of 178 individuals. Trasarti et al. [19] use GPS data to build the mobility profiles of 2107 individuals and match them based on the similarities of their mobility profiles; they also apply their algorithms to a GSM-like data set, which they synthesize by reducing the size of their GPS data set.

Past work on Carpooling has be focused mainly in characterizing the behavior of carpoolers, identifying the individuals who are more likely to carpool and explaining what are the main factors that affect their decision on whether to use carpooling or not [16]. Also, the question whether ride-sharing can reduce congestion has been asked before [20]. But, the authors of [20] assumed uniform distribution of home and work locations and concluded that ride-sharing has little potential for congestion reduction. On the contrary, we make no assumption about the distribution of home and work locations in a city, but infer such information from a Call Description Dataset, and show that ride-sharing has a lot of potential for reducing the number of cars from the streets of a city.

7. CONCLUSION

In this paper, we used CDR data to derive an upper bound for the potential of ride-sharing. The results indicate that there exists huge potential in densely populated urban areas, such as the city of Madrid in Spain. This motivates working on the technological challenges involved in facilitating car sharing. In future work, we plan to (i) extend our study in other cities, (ii) take additional aspects into account, such as the structure of the call graph, (iii) obtain additional information for ground truth and (iv) deploy an actual system that mines CDR and facilitates car sharing.

Table 1: Summary of results, $\delta = 0.6$ km

drivers (%)	δ (km)	τ (min)	σ (min)	`EndPoints RS` (%)	`EnRoute RS` (%)
100	0.6	–	–	**56**	**69**
100	0.6	10	10	**41**	**60**
100	0.6	10	20	**26**	**50**
60	0.6	–	–	**52**	**67**
60	0.6	10	10	**35**	**56**
60	0.6	10	20	**20**	**46**
30	0.6	–	–	**44**	**63**
30	0.6	10	10	**27**	**50**
30	0.6	10	20	**13**	**40**

The first column indicate the % of users owning a car, the second indicates the distance tolerance, the third parameter indicates the time tolerance, the fourth the time spread and the last two columns show the success of carpooling for `EndPoints RS` and `EnRoute RS`.

8. REFERENCES

[1] Instituto de estadistica de la comunidad de madrid: http://www.madrid.org/iestadis/.

[2] The true cost of a car over its lifetime: http://www.doughroller.net/smart-spending/true-cost-of-a-car-over-its-lifetime.

[3] K. Cechlárová and T. Fleiner. On a generalization of the stable roommates problem. *ACM Trans. Algorithms*, 1(1):143–156, July 2005.

[4] Eunjoon Cho, S.A. Myers, and J. Leskovec. Friendship and mobility: user movement in location-based social networks. In *Proc. ACM SIGKDD*. ACM, 2011.

[5] Nigel Davies, Manfred Lau, Chris Speed, Tom Cherrett, Janet Dickinson, and Sarah Norgate. Sixth sense transport: challenges in supporting flexible time travel. In *Proceedings of the Twelfth Workshop on Mobile Computing Systems and Applications*, HotMobile '12, pages 8:1–8:6. ACM, 2012.

[6] N. Eagle, A. Pentland, and D. Lazer. Inferring friendship network structure by using mobile phone data. *Proceedings of the National Academy of Sciences of the United States of America*, 106(36):15274–8, September 2009.

[7] M. Ficek and L. Kencl. Inter-Call Mobility Model: A Spatio-temporal Refinement of Call Data Records Using a Gaussian Mixture Model. In *Proc. Infcocom*, 2012.

[8] M. C. González, C. A. Hidalgo, and A. L. Barabási. Understanding individual human mobility patterns. *Nature*, 453(7196):779–82, June 2008.

[9] W. He, D. Li, T. Zhang, L. An, M. Guo, and G. Chen. Mining regular routes from gps data for ridesharing recommendations. In *UrbComp*. ACM, 2012.

[10] S. Isaacman, R. Becker, R. Cáceres, S. Kobourov, M. Martonosi, J. Rowland, and A. Varshavsky. Identifying Important Places in People's Lives from Cellular Network Data. In *Proc. Pervasive*, June 2011.

[11] S. Isaacman, R. Becker, R. Caceres, S. Kobourov, M. Martonosi, J. Rowland, and A. Varshavsky. Ranges of human mobility in Los Angeles and New York. In *Proc. MUCS*, March 2011.

[12] S. Isaacman, R. Becker, R. Cáceres, S. Kobourov, J. Rowland, and A. Varshavsky. A tale of two cities. In *Proc. HotMobile*, February 2010.

[13] S. Isaacman, R. Becker, R. Cáceres, M. Martonosi, J. Rowland, A. Varshavsky, and W. Willinger. Human Mobility Modeling at Metropolitan Scales. In *Proc. MobiSys*, June 2012.

[14] Madhukar R. Korupolu, C. Greg Plaxton, and Rajmohan Rajaraman. Analysis of a local search heuristic for facility location problems. In *Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 1998.

[15] V. Soto and E. Frias-Martinez. Automated Land Use Identification using Cell-Phone Records. In *HotPlanet*, 2011.

[16] R.F. Teal. Carpooling: Who, how and why. *Transportation Research*, 21A(3):203–214, 1987.

[17] Arvind Thiagarajan, Lenin S. Ravindranath, Hari Balakrishnan, Samuel Madden, and Lewis Girod. Accurate, low-energy trajectory mapping for mobile devices. In *8th USENIX Symp. on Networked Systems Design and Implementation (NSDI)*, Boston, MA, March 2011.

[18] Arvind Thiagarajan, Lenin S. Ravindranath, Katrina LaCurts, Sivan Toledo, Jakob Eriksson, Samuel Madden, and Hari Balakrishnan. Vtrack: Accurate, energy-aware traffic delay estimation using mobile phones. In *7th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, Berkeley, CA, November 2009.

[19] R. Trasarti, F. Pinelli, M. Nanni, and F. Giannotti. Mining mobility user profiles for car pooling. In *Proc. UrbComp*. ACM, 2011.

[20] H.-S. J. Tsao and D. Lin. Spatial and temporal factors in estimating the potential of ride-sharing for demand reduction. California PATH Research Report, UCBITS-PRR-99-2, 1999.