Seamless TCP Migration on Smartphones without Network Support

Ahmad Rahmati, *Member*, *IEEE*, Clayton Shepard, *Student Member*, *IEEE*, Chad C. Tossell, Lin Zhong, *Member*, *IEEE*, Philip Kortum, Angela Nicoara, *Member*, *IEEE*, and Jatinder Singh, *Member*, *IEEE*

Abstract—*Is it possible to migrate TCP/IP flows between different networks on modern mobile devices without infrastructure support or protocol changes*? To answer this question, we make three research contributions: 1) We report a comprehensive characterization of IP traffic on 27 iPhone 3GS users for three months. 2) Driven by these findings, we devise two simple, effective, and easily deployable *system mechanisms* to support seamless flow migration without network support, and extensively evaluate their effectiveness using our field collected traces of real-life usage. *Wait-n-Migrate* leverages the fact that most flows are short lived. It establishes new flows on newly available networks but allows preexisting flows on the old network to terminate naturally. *Resumption Agent* takes advantage of the resumption functionality of modern protocols to securely resume flows without application intervention. Combined, they provide an unprecedented opportunity to immediately deploy policies that leverage multiple networks to improve the performance, efficiency, and connectivity of mobile devices. 3) We report an iPhone-based implementation of these system mechanisms and demonstrate their overhead to be negligible. Furthermore, we employ a sample switching policy, *AutoSwitch*, to demonstrate their performance. Through traces and field measurements, we show that AutoSwitch reduces user disruptions by an order of magnitude.

Index Terms—Mobile computing, network architecture and design, user/machine systems

1 Introduction

Modern mobile devices have access to multiple networks. Not only do they have multiple network interfaces, such as cellular and Wi-Fi, but also a single interface may access multiple networks, such as Wi-Fi hotspots from different providers. Over time, the networks available to a mobile device and their qualities vary greatly, for example, as the user moves. A large body of recent work attests to the value of properly switching between networks [1], [2] or aggregating them [3], [4], [5]. Switching between networks can significantly improve the performance [6], [7], energy efficiency [1], [8], and connectivity [9] of mobile Internet. In this work, we focus not on *policies*, i.e., determining when to switch, but *mechanisms* to enable switching and/or aggregating networks on smartphones.

The key to switching between networks or aggregating them is to change the network for existing flows without disrupting their corresponding applications. Brute-force switching between networks, where one network is simply disabled and another enabled, may lead to undesirable disruptions, as our own experience corroborates and as confirmed by our user study. Solutions to this problem are available in the name of *handoff*. Some require infrastructure or home agent support, for example, cellular handoff, connection gateway, and Mobile IP, which incur extra operating expenses and additional latency, hampering adoption and reducing the performance of both interactive [10] and noninteractive [11] applications. Others require changing the TCP/IP protocol, which has been shown to be practically very difficult. Not surprisingly, no automatic switching or aggregating solutions have been widely deployed in practice.

The important question this paper addresses is the following: On modern mobile devices, is it possible to seamlessly migrate TCP/IP flows between different networks without changes to preexisting applications, infrastructure, and protocols? Toward answering this question, this paper presents three research contributions.

First, we report a comprehensive characterization of network traffic on smartphones using three-month traces collected from 27 iPhone 3GS users. The characterization provides key insights into the motivation and rationale of our mechanisms. In particular, we have found that network flows are typically short-lived and utilize standard protocols, long-lived flows are often predictable and automatically reconnect upon disruption, and that there are few concurrent flows during interactive usage.

Second, we present and extensively evaluate two novel system mechanisms, implemented in a smartphone, with the objective of migrating flows between networks without network support and without disruption to the user. The

- *A. Rahmati is with Broadcom Corporation, 190 Mathilda Pl., Sunnyvale, CA 94086. E-mail: ahmad@rahmati.com.*
- *C. Shepard, L. Zhong, and P. Kortum are with Rice University, Houston, TX 77005. E-mail: {cws, lzhong, pkortum}@rice.edu.*
- *C.C. Tossell is with the USAF Research Laboratory, Wright-Patterson Air Force Base, 2620 Q Street, Bldg. 852, OH 45433. E-mail: chadtossell@yahoo.com.*
- *A. Nicoara is with the Deutsche Telekom Silicon Valley Innovation Center, 295 N. Bernardo Ave., Suite 200, Mountain View, CA 94043. E-mail: angela.nicoara@telekom.com.*
- *J. Singh is with Stanford University, 215 Packard Building, 350 Serra Mall, Stanford, CA 94305. E-mail: jatinder@stanford.edu.*

Manuscript received 17 June 2011; revised 25 Jan. 2012; accepted 27 Dec. 2012; published online 10 Jan. 2013.
For information on obtaining reprints of this article, please send e-mail to: tmc@computer.org, and reference IEEECS Log Number TMC-2011-06-0324. Digital Object Identifier no. 10.1109/TMC.2013.8.

first mechanism, *Wait-n-Migrate*, takes advantage of the fact that TCP flows are short lived. It establishes new flows on the new network, but waits for the preexisting flows on the old network to terminate normally, up to a specific wait-time set by the migration policy. The *wait-time* determines the tradeoff between faster switching and fewer dropped flows. The second mechanism, *Resumption Agent*, leverages the resume function in modern servers and resumes a flow from wherever it was disrupted, in a manner transparent to applications and with negligible overhead. Resumption Agent provides automatic resuming capabilities to all preexisting applications, in an application agnostic manner. Based on our traces, we show that Wait-n-Migrate can successfully migrate 90 and 95 percent of web flows without interruption, for wait-times of 10 and 100 seconds, respectively. With the addition of Resumption Agent, we show that for web flows that support resuming, we can virtually eliminate disruptions when switching between networks.

Third, we report an efficient implementation of the Wait-n-Migrate and Resumption Agent mechanisms on the iPhone platform, and show that their overhead is negligible. Based on the two system mechanisms, we further implement a sample network interface switching policy, AutoSwitch. AutoSwitch uses Wait-n-Migrate and Resumption Agent to offload data from cellular to Wi-Fi as much as possible, with minimum disruptions to the user. AutoSwitch using Wait-n-Migrate alone achieves over one order of magnitude reduction in the number of disruptions in our real-life traces, and from over 40 percent to well under 10 percent for 100 KB transfers while driving. Furthermore, when the content supports resuming, disruptions are almost entirely eliminated with the addition of Resumption Agent.

The rest of this paper is organized as follows: In Section 2, we present a motivational user study to show that brute-force network switching is unacceptable to users, and then discuss related work. In Section 3, we present the characterization of network traffic on 27 iPhone 3GS users and provide insight to the characteristics of network flows on modern smartphones. Based on these findings, in Section 4, we present the design and trace-based evaluation of Wait-n-Migrate and Resumption Agent. In Section 5, we report their implementation on iPhone and evaluate their performance impact. In Section 6, we present an example application, AutoSwitch, of the resulting seamless flow migration without network support. Finally, we discuss methods to further enhance our mechanisms for increased performance in Section 7, and conclude in Section 8.

2 Background

2.1 Consequences of Brute-Force Switching

Without network support, smartphones switch between networks (e.g., cellular and Wi-Fi) in a brute-force manner: They terminate all flows on the old network and enable the new network. This behavior is shared across all the three major smartphone platforms we studied: iOS, Android, and Windows Mobile.[1] It is then up to the application, or often the user, to detect the disruption and retry over the new network. This brute-force switch introduces disruptions to interactive sessions. According to our personal experience, network disruption is noticeably annoying, and particularly prevalent for large webpages or during poor connectivity. To better understand the usability impact of network disruption (e.g., as will be experienced due to brute-force switching), we performed a formal user study with 10 participants from the Rice student community who already used Internet-ready smartphones. The study had equal males and females, and four participants with non-engineering backgrounds.

Our study consisted of two parts. The first part asked the users to open a copy of a regular news website cached on our server for consistency. We then asked users to perform a number of text identification tasks on three individual pages. The participants were later directed to a cached copy of a mobile news search engine, where they were asked to identify several stories and their sources. During the study, our server automatically disrupted the data flow for the first load of three of the five page loads. The users had to refresh their browser to completely load each page. This simulated the impact of a brute-force migration. Participants were free to either use their own phones or our iPhone for the purpose of this study.

For the second part, we interviewed the participants to assess their browsing experience, including several questions on a 1-5 Likert scale (agree-disagree), and several open-ended questions. All 10 participants agreed or somewhat agreed that disruptions are an *annoying experience*. Interestingly, all 10 also agreed or somewhat agreed that they *have had similar experiences prior*, and that they *typically refresh a page that has failed to completely load*.

While the participants' prior network disruption experiences are typically due to bad connectivity, brute-force network switching will cause similarly unwanted and annoying disruptions.

During the open-ended question sessions, when asked whether they have experienced this phenomenon more often in specific websites, nine of 10 mentioned that they experience it more frequently with larger transfers, for example, mentioning pages that are as "heavier" or "with lots of graphics." When asked whether they have experienced this phenomenon more often in specific conditions, eight of 10 correctly identified that they experience it more frequently during one or more network conditions (e.g., low signal, moving). We can see that even without intentional network switching, users are subject to unwanted and annoying network disruptions. This further motivates our AutoSwitch policy, as presented in Section 6.

While our user study was conducted with a small number of participants ($n = 10$), considering the high confidence intervals, our findings are expected to be true with the majority of user populations similar to our participants. For example, the 90 percent Agresti-Coull confidence interval [12] for eight and 10 positive answers out of 10 are (0.52, 0.91) and (0.66, 1), respectively, i.e., there is a 90 percent chance that the statistics for the population falls in those intervals.

In summary, we confirmed that network disruptions annoy users. We also found that typical users have

1. The only exception was iOS and only when switching from cellular to Wi-Fi, where it keeps existing connections indefinitely on their original interface.

extensive experience with network disruptions, and have even figured out the conditions in which they often occur. A successful solution to for network disruptions must not blatantly change the user experience or discard the partially received content. These findings motivate and assist both the design of our mechanisms and our example application, AutoSwitch.

2.2 Related Work

TCP/IP traditionally lacks built-in support for switching between multiple networks (handoff) or aggregating their throughput (multihoming). Therefore, there exists a body of research on providing session continuity [13] between different networks, i.e., maintaining the same IP address while moving between networks. Current solutions for session continuity fall into three categories. First is to have one network as the slave to a master network, where all traffic is directed through the latter [14], as in Virtual AP. However, this requires unified management of the networks, increases traffic on the master network, and increases latency. The second category of solutions utilize a mobility gateway in the infrastructure [15], [16], to act as a proxy between a mobile device and the Internet. For example, such gateways have been employed for switching between interfaces (Wiffler [6]), for multihoming [3], [4], [17], and for striping [5], [15], [18], [19], [20], [21], [22], [23], [24]. However, routing all flows through a fixed gateway can increase the connection latency. The third category of solutions modify or extend the TCP/IP protocol support for mobility, for example, by adding explicit support, as in [25], [26], or through Mobile IP [27], [28], [29], where a home router or agent handles mobility and packet forwarding. However, the extra forwarding increases the traffic on the home agent and more importantly, the extra distance traveled by packets increases the connection latency, which is known to be a major bottleneck for mobile Internet performance [10], [11]. Mobile IPv6 eliminates the need for a specific foreign agent, but in return requires individual mobile nodes to perform the forwarding operations, with similar drawbacks.

While all three categories of solutions discussed above are designed to successfully migrate all network flows for every network switch, they require additional infrastructure or network support in addition to device modifications, and thus are not immediately deployable. Those that have begun deployment suffer from limited or unsuccessful adoption. Furthermore, solutions that depend on a forwarding gateway or Mobile IP increase network latency. Latency is known to be a major bottleneck in mobile Internet performance, for both interactive applications [10] and noninteractive (browser) applications [11].

In contrast to these existing techniques, our mechanisms require no changes to applications, infrastructure, or network protocols, and induce no additional connection latency. Consequently, our mechanisms allow the immediate deployment of system policies that leverage multiple networks. The tradeoff of our mechanisms is that a small fraction of network flows are abruptly terminated. However, we show that this small fraction of terminated flows is less than the number of flows that are terminated due to regular phone usage in changing network conditions, for example, when moving.

The practical value and applicability of this work is further highlighted by recent work using our mechanisms. One such example is MultiNets [30], which utilizes our Wait-and-Migrate mechanism to switch between cellular and Wi-Fi networks, to save energy, offload data from cellular networks, and/or improve performance.

There are two solutions related to *Resumption Agent*. Resuming static content is typically supported by download managers such as *wget*. Yet, most other applications (e.g., browsers) lack resume functionality. In contrast, Resumption Agent is an application agnostic solution that provides automatic resuming capabilities to all preexisting applications. Snoeren et al. [31] supported resumption through a client agent for the purpose of failover between replica servers, while keeping servers largely unchanged. However, Resumption Agent does require replica servers, and is compatible with most existing servers, while handling the challenges of dynamic content and secure HTTPS connections.

Recently Alperovich and Noble [32] have proposed to improve Wi-Fi performance for PC clients by switching and balancing connections between multiple Wi-Fi access points (APs), for example, as enabled through Virtual Wi-Fi [33]. They also retain preexisting connections on their original AP, while assigning new connections to new APs. Yet, our work focuses not on load balancing, but evaluates mechanisms for switching between heterogeneous networks on smartphones. We go beyond retaining preexisting connections by addressing long-lived flows and supporting preexisting applications on mobile phones.

There has also been several studies addressing smartphone usage and network traffic characteristics [34], [35]. Our contribution in traffic characterization complements these works. In addition, using our traces, we are able to evaluate the efficacy of our proposed mechanisms for network migration by providing detailed analysis of traffic protocols, flow length and concurrency, and the active application concurrent to the flows.

Finally, we note that we have previously presented the initial ideas along with partial results as a MobiCom 2010 poster [36], and later in a more complete manner as a Rice University Technical Report [37].

3 Network Flow Characterization

A thorough understanding of the characteristics of network flows on modern mobile devices is critical to the seamless migration of flows. We next report a first-of-its-kind study based on detailed network flow traces from 27 iPhone 3GS users. The characterization provides key insights for our design, as described in Section 4.

3.1 iPhone Field Trace Collection

We gathered real-life network traces from 27 iPhone 3GS users over the course of three months by installing logging software we developed, called LiveLab [38]. The phones were running the iOS 3.x operating system throughout the study, the latest version available at the beginning of the study. We chose the iPhone 3GS because at the beginning of

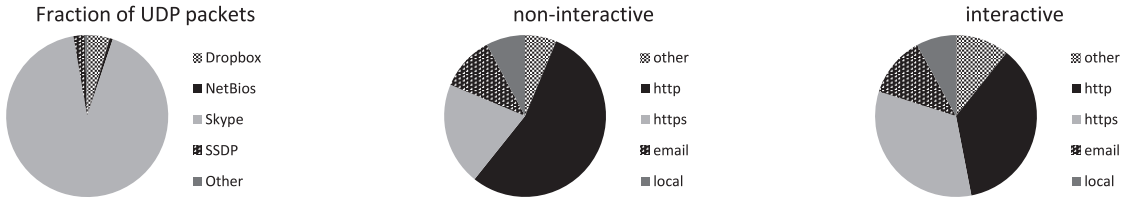

Fig. 1. Fraction of packets according to application for UDP packets (Left). Fraction of TCP flows for each application type (Center: noninteractive sessions. Right: interactive sessions, i.e., phone display was on).

the study, it represented the cutting edge of smartphone design, accounting for 55 percent of all mobile internet traffic in the US as of October 2009 [39]. Additionally, iPhone users have access to the largest number of third-party applications, with over 300,000 officially released apps as of October, 2010.

Whenever the phone's CPU is not asleep, LiveLab records TCP network connection statistics every 2 seconds by running the *netstat* tool. Its output is similar to the same tool available on Windows and Linux/Unix platforms. Moreover, LiveLab records the application being used and the display status in real time, and Wi-Fi signal strength for the currently connected AP and all visible APs every 2 seconds and 15 minutes, respectively. Finally, it recorded the complete packet headers for three of the participants over one month, to gauge the data flow over UDP. We refrained from deploying this packet-level logging for longer time or more users due to its overhead. The data are recorded on the phones, and is transferred nightly to our servers in a secure fashion.

While our participants were not recruited to accurately represent the vast mobile user population, the collected data provide an unprecedentedly detailed look into the connectivity on contemporary mobile devices.

3.2 Focus on TCP Flows

The packet-level logging data show that out of the three common IP protocols in use TCP, UDP, and ICMP. ICMP packets are typically not used by interactive applications, but by devices for diagnostics, device discovery, and error messages specific to each network. Therefore, for the purpose of switching between networks, they can be safely ignored. TCP and UDP account for 93 and 7 percent of all remaining packets, respectively. TCP flows present the main challenge toward flow migration. While we will examine TCP flows in detail later, we first discuss UDP flows. We analyze the UDP flows based on port numbers, and further corroborate this analysis with the applications currently being used. We have found the following services and applications utilize UDP on the phones (Fig. 1):

- Skype (92 percent) uses UDP ports 12340 and 20515.
- Dropbox (4 percent) uses UDP broadcast on port 17500.
- Simple Service Discovery Protocol (SSDP) (2 percent) is used to advertise and discover network services.
- NetBIOS (1 percent) for local area network device discovery and networking.
- Other (<1 percent) such as NAT Port Mapping.

With the exception of Skype, all of these are network and discovery services and specific to a particular network. Therefore, we will ignore them for the purpose of switching between networks, similar to ICMP traffic.

For Skype, we have found that as long as the primary interface in the system routing table is correctly updated, for example, as is the case with our mechanisms or when the user manually enables or disables Wi-Fi, Skype switches to the new network for both its TCP and UDP connections, without dropping a call and with only a very short period ($\sim$1 sec or less) of muting in the audio. However, if the system is unaware of the disruption (e.g., moving out of Wi-Fi coverage), Skype will drop the call. This highlights the importance of a systemwide notification of a network change, instead of simply losing connectivity, for the benefit of applications that can handle disruptions gracefully. AutoSwitch, described later in Section 6, achieves this through updating the routing table.

Therefore, for the remainder of this paper, we will focus on TCP flows. Using the port number of the server, we divide external TCP flows into three categories:

- *Web (HTTP: 80, HTTPS: 443)*. These are used not only by the browser, but also by a number of native applications that utilize web services or a built-in browser.
- *E-mail (IMAP: 143, 993, POP3: 110, 995, SMTP: 25, 465)*. These are used by the native e-mail client, and will not include e-mail accessed through the browser.
- *Other*. All other applications and services.

Fig. 1 shows the fraction of TCP flows utilized for each application during both interactive and noninteractive usage. We use the display status (on) as an indicator of the phone is being used interactively. We can see that more than three quarters of TCP streams are web flows, highlighting the importance of handling them properly. We also separate and ignore local (loopback) flows that reside only on the phone.

Figs. 2 and 3 show the probability density function (PDF) of the number of flows and the cumulative distribution function (CDF) of flow lifetimes, respectively, according to TCP port for both interactive and noninteractive sessions whenever the phone's CPU was running. We can see that flows have similar characteristics during interactive and noninteractive usage, yet, on average, flows during interactive usage have slightly shorter lifetimes. In the following sections, we will study them in further detail, according to application use.

3.3 Flow Concurrency

While analyzing the LiveLab data, we were surprised to discover that there are few concurrent flows on the iPhone platform, with negligible difference between interactive and

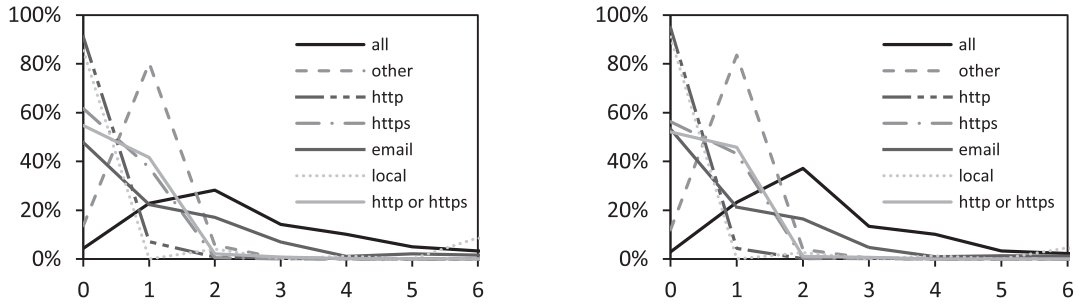

Fig. 2. We observed few concurrent TCP flows (median $= 2$), and noninteractive and interactive usage were similar. PDF of the number of concurrent TCP flows for different TCP ports, average among all users. (Left: noninteractive sessions. Right: interactive sessions.)

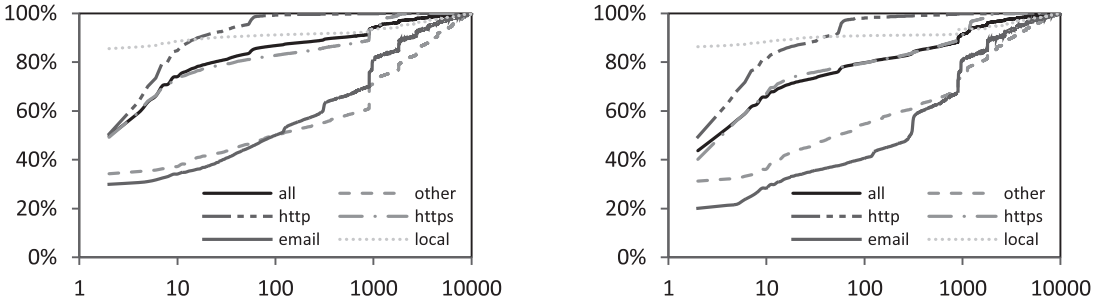

Fig. 3. Flows during interactive usage have slightly shorter lifetimes. CDF of TCP flow lifetimes in seconds, based on TCP port, average among all users. (Left: noninteractive sessions. Right: interactive sessions.)

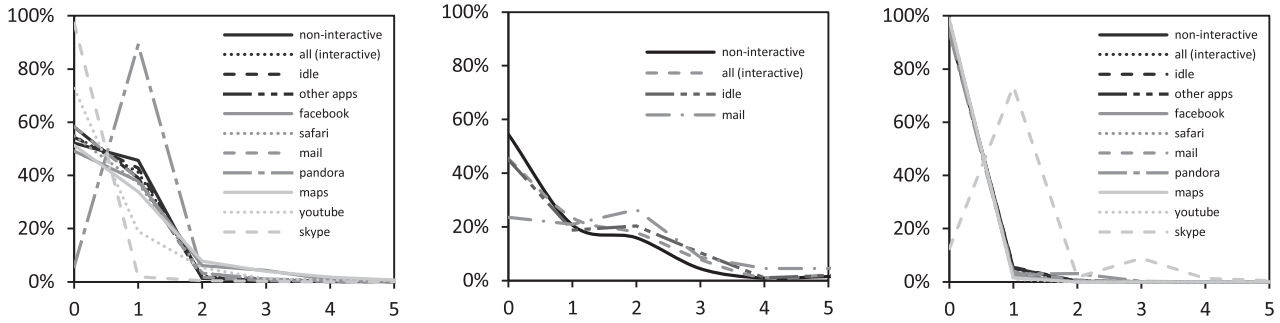

Fig. 4. PDF of the number of TCP flows on different port numbers when running different Internet applications, excluding the push service. Left: web ports. Center: e-mail ports. Right: other ports.

noninteractive sessions. The median number of flows was 2. However, there almost always exists one particular flow, 97 percent of the time that the phone is awake. We have identified that flow as *Apple's push notification service, on port 5223*. Fig. 4 shows the PDF of number of concurrent TCP flows, excluding the Apple Push service, whenever the phone's CPU was running and for the port types presented in Section 3.2 (web, e-mail, other). We identified the top seven applications that require Internet access using the data from our field study, which include Pandora (music streaming) and Skype (instant messaging, voice over IP). These applications account for over 95 percent of interactive phone Internet use. Noninteractive usage, including when the display was off, idle time, when the home screen was displayed are presented separately. Other applications, including those that do not require internet connectivity, are clustered together as others. For e-mail and other ports, we display only the applications that we have determined to use those ports.

We can see that even when running internet enabled applications, the phone is rarely engaged in multiple TCP flows simultaneously. The small numbers of simultaneous TCP flows shows that *for web applications on mobile phones, multihoming mechanisms (i.e., no-striping) are effective for at most 20 percent of flows*, as the other 80 percent of times when a web flow exists, it is a single flow. However, we expect more simultaneous flows as more applications and services become available on mobile devices. The e-mail client, while not typically data intensive, presents an exception, as it regularly uses multiple flows.

3.4 Flow Lifetime

We have found that most interactive flows on the phone were short lived, and it is often possible to automatically predict the duration of flows based on application and port numbers. We measure the flow lifetime without including the connection/tear down phase (e.g., wait_fin). We note that some flows may not terminate naturally, but instead may be disconnected due to network signal issues (or, e.g., the user manually switching Wi-Fi off). We are unable to accurately distinguish between a naturally terminated flow and one that is terminated due to connectivity issues. However, a flow that is disconnected due to connectivity issues typically has a longer lifetime than it originally would,

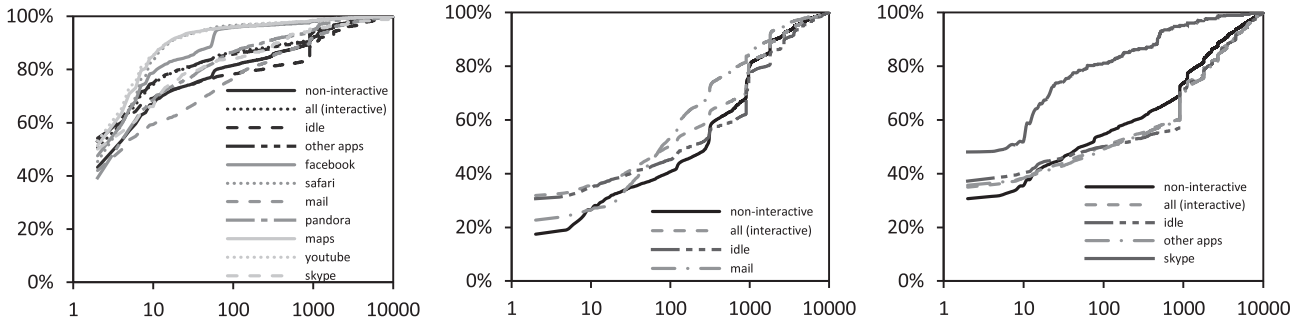

Fig. 5. CDF of TCP flow lifetimes (seconds) on different port numbers, based on active application. Left: web ports. Center: e-mail ports, Right: other ports.

as disconnections are detected only after a preset time-out has passed with no data transfer. Consequently, connection issues result in a slight overpresentation of flow lifetimes, but we believe the effect is negligible, as only small proportion of flows is terminated due to connectivity issues.

Our logs show a wide variation in the lifetime of TCP flows on the experimental phones, in particular between interactive and noninteractive usage sessions. Fig. 5 shows, on average among our participants, the CDF of TCP session lengths for different TCP ports and different active applications.

Our first finding is that *most flows are short lived*. In fact, 50 and 44 percent of flows for noninteractive and interactive sessions, respectively, are ~2 seconds or less. In turn, *this limits the effectiveness of power saving schemes which rely on long-lived downloads*, such as CatNap [40].

Our second finding is that it is possible to predict flow lengths based on active application and port, i.e., the distribution of flow lifetimes varies significantly based on TCP port, active application, and whether the phone is being used interactively. For example, as shown in Fig. 5, the fraction of short-lived e-mail flows (i.e., IMAP, SMTP, POP3) is much lower: 30 and 20 percent for noninteractive and interactive sessions, respectively. Similarly, the Apple Push service is known to be long lived. On the other hand, as shown in Fig. 5, TCP flows during web browser sessions were shorter than average. We will later see how these findings are important for our switching mechanisms.

3.4.1 Long-Lived Nonstandard TCP Flows

We next consider long-lived flows that use nonstandard protocols, other than web, ftp, and e-mail. Such flows are difficult, if not impossible, to migrate without network support. However, a close examination reveals that such flows usually do not require migration support at all.

First, long-lived TCP flows based on closed application protocols are usually from background, noninteractive applications. Therefore, while their disruption or brute-force migration may, for example, slightly delay an update, they will rarely be noticeable to users.

More importantly, the handful of applications that do utilize long-lived nonstandard protocols already provide support to migration in various forms because *application developers anticipate the possibility of disruptions of long-lived flows*. For example, applications such as Push notifications, Twidroid, and many instant messaging applications are designed to gracefully and automatically reestablish a connection after being disconnected. Another example, Pandora, a common Internet radio streaming application, and the only one that appeared in our participants' list of top 25 applications, skips the unbuffered part of the current song, i.e., at most suffer skipping part of a track.

3.5 Background Applications

While the iPhone 3GS we used in the study was the state-of-the-art phone at its time, it lacked official support of multitasking for third-party applications as of OS 3.x. Note that background operation is supported for some native applications, for example, the e-mail client. We note that Android and the newly released iPhone iOS 4.0 allow background applications, for example, Skype and Pandora, to access data networks [41]. This, alongside the increasing processing power and memory of phones, suggests an increase in the usage of background capable applications (e.g., instant messaging, Twidroid). Therefore, in the future, we would expect an increase in the simultaneous network flows, from those shown in Figs. 2 and 4.

Yet, assuming connection length distributions remain unchanged, increased multitasking will not affect the usability and effectiveness of the general case of Wait-n-Migrate nor the Resumption Agent. Indeed, assuming the device can remain connected to two networks simultaneously, flows from multiple simultaneous applications will not affect each other and we can consider each application independently. Therefore, we believe our results are valuable on newer iOS versions and other operating systems with multitasking support.

4 Migration without Network Support

Based on the findings from Section 3, for the objective of migrating network flows between networks, we focus on seamlessly migrating short-lived flows or flows using standard protocols such as HTTP and FTP. We provide two novel and complementary mechanisms for migrating such flows without changing preexisting applications, infrastructure and protocols, and with minimal disruption to the user. We envision that in most systems, Wait-n-Migrate will be used primarily, and Resumption Agent will be used to migrate flows that were not successfully migrated by Wait-n-Migrate.

4.1 Wait-n-Migrate

Our first method leverages the fact that most flows are short lived, as seen in Section 3. Wait-n-Migrate typically requires

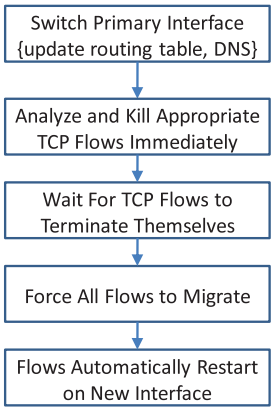

Fig. 6. Flowchart for Wait-n-Migrate.

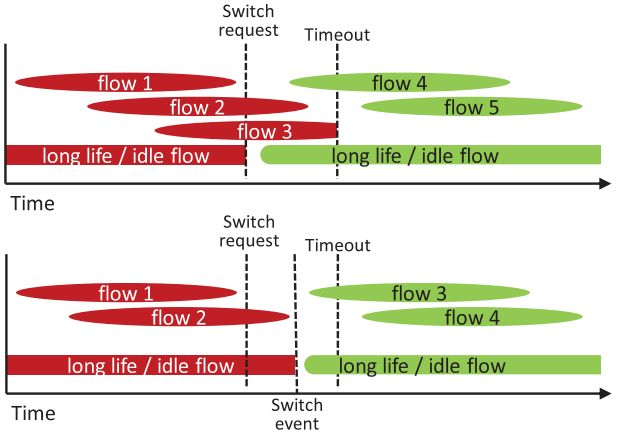

Fig. 7. Wait-n-Migrate operation (Top), and the special case without requiring simultaneous connectivity (bottom).

the device to be able to connect to multiple networks simultaneously. This may be through multiple interfaces (e.g., 3G and Wi-Fi) or through one interface (e.g., multiple Wi-Fi networks through Virtual Wi-Fi [33]).

To migrate one or more flows between two networks, Wait-n-Migrate operates as follows (Fig. 6). First it enables both networks so the system has simultaneous connectivity to both. Second it ensures all new flows are created on the new network. Then, it waits for the flows on old network to terminate naturally, up to a specific wait-time (Fig. 7). The wait-time for each flow is a parameter determined by the particular migration policy and can be set according to application, bandwidth, and power considerations, and may be adaptive according to flow characteristics presented in Section 3. A flow is assumed to be successfully migrated when it terminates naturally by the wait-time.

Different wait-time values can be used for switching to different networks. For example, when the system policy requests a network switch to a slower or less efficient network, for example, to assure connectivity, the wait-time can be set to infinite, i.e., until losing connectivity. On the other hand, when switching back to the faster/more efficient network, a shorter wait-time should be used. Finally, if there are no remaining flows on the old network, the system can disable or power it off altogether.

For systems that can be connected to only one network at a time (i.e., without simultaneous network connectivity), a special case of Wait-n-Migrate can be used. This special case takes advantage of the fact that there are few simultaneous TCP flows. It monitors TCP flows and attempts to choose the best moment to switch within a specifically allowed time range, to minimize disruptions. This is possible through the statistical properties of TCP flows, as presented in Section 3.

Finally, Wait-n-Migrate can employ flow lifetime prediction to further improve its effectiveness and efficiency. Wait-n-Migrate does not interfere with short-lived flows to avoid user disruption. However, for flows that are known to be highly likely to live beyond the wait-time, for example, based on the findings in Section 3, Wait-n-Migrate can terminate them immediately. For example, we already know that several types of flows are long lived, for example, Push notifications and idle e-mail flows. If the device is switching to a faster or more energy-efficient network, Wait-n-Migrate can terminate such flows immediately, thus improving performance.

4.2 Resumption Agent

Our second method, Resumption Agent, leverages the fact that many interactive applications use standard application layer protocols such as HTTP, HTTPS, as highlighted in Section 3.2, and that most servers for these protocols support resume. Resumption Agent is a locally run proxy that enables flow migration for most such flows. It provides a safety net to reduce the user impact of network switching when Wait-n-Migrate terminates a flow for migration. With Resumption Agent, Wait-n-Migrate can be more aggressive in migrating flows and, therefore, allow for faster switching.

Resumption Agent can support any application protocol that allows for resuming from an arbitrary location within a data transfer. Several key standard application-layer protocols, including HTTP and FTP, provide adequate support for resumption of a terminated transfer. For example, the HTTP standard, from version 1.1 onwards (1996), supports specifying a *range* when requesting a webpage. The FTP standard also supports resuming via the *rest* command. Standard e-mail protocols (e.g., IMAP, POP, and SMTP) can also be restarted from the beginning of any e-mail, or any individual attachment in the case of IMAP.

Resumption Agent works as follows: It requires a background service running only on the device itself, which acts as a proxy, and modifies the phone settings so that applications use this proxy to connect to the internet. If a flow is disconnected prematurely, Resumption Agent automatically resumes the transfer from where the flow was cut off. Therefore, when a flow needs to migrate to a new network, it can be terminated on the old network and resumed on the new network in transparent manner to the application.

We conjecture that Resumption Agent can employ flow lifetime prediction to further improve its effectiveness and efficiency. For web flows, their sizes are typically know at the beginning of the transfer, through the HTTP header response *Content-Length*. Therefore, if Resumption Agent is used in conjunction with Wait-n-Migrate, flows that are expected to last beyond the wait-time can be killed and resumed immediately by Resumption Agent.

We note that download managers, such as *wget*, support the automatic resuming of static content. Yet, they are unable to handle the challenge of unsupported content

(as discussed below). More importantly, web browsers (on both PCs and phones), and most other applications (e.g., the iPhone YouTube application) lack automatic resuming functionality. In contrast, Resumption Agent is application agnostic and appears as a regular proxy server to applications, thus providing a system level solution for all preexisting applications. Moreover, Resumption Agent can handle network migration and two nontrivial challenges to Resumption Agent for web flows, posed by unsupported content and encrypted HTTPS flows. We next discuss them and present our solutions.

4.2.1 Unsupported Content

There are three groups of content that cannot be resumed in the middle of the transfer:

1. The first group includes content that does not allow resuming. For example, some servers may ignore HTTP Range requests altogether or for specific content, such as small transfers, or chunk encoded data (the size of the data is not known beforehand). In this case, the transfer, if interrupted, must be restarted from the beginning, resulting in a second and unnecessary transfer of the initial portion, which the Resumption Agent will ignore.
2. The second group is content uploads, usually using HTTP POST, in which there is always the risk of repeating an action, for example, a purchase. In such cases, such as when the user refreshes a page with POST content, web browsers present the user with a warning. Resumption Agent uses the same behavior and will avoid automatically resuming such a transfer if it is disconnected.
3. The third group is dynamic content that changes significantly for every reload. Resumption Agent deals with dynamic content using two methods. First, the HTTP headers *Pragma:no-cache* and *Cache-Control:no-cache* in the request and response headers, respectively, indicate dynamic content, as the prevent proxies and other web servers from caching the content. Thus, if Resumption Agent sees these tags, it can abstain from automatically resuming a failed transfer. Second, to support dynamic content that does not provide hints in the headers, Resumption Agent always resumes from a preset length prior to the disruption. It then compares the overlapping sections. If the overlapping sections are identical, Resumption Agent will simply continue with the resume. If the overlapping sections become identical after applying a small offset to the data, for example, to account for a slightly smaller or larger dynamic advertisement content, it will correct the offset and can continue with the resume. Only if the overlapping sections are not identical even after applying an offset, will Resumption Agent abort the resume and the transfer will fail.

4.2.2 Encrypted HTTPS Flows

A greater challenge comes from HTTPS, as it is impossible for a regular proxy to directly inspect its contents, which is end-to-end encrypted by SSL. Indeed, when an application wants to connect to a HTTPS server through a typical proxy, it sends a CONNECT command to the proxy. The proxy then creates a tunnel to the requested server, without touching the transferred content. Such end-to-end encryption would make it impossible to analyze the data, necessary for transparently resuming or striping transfers.

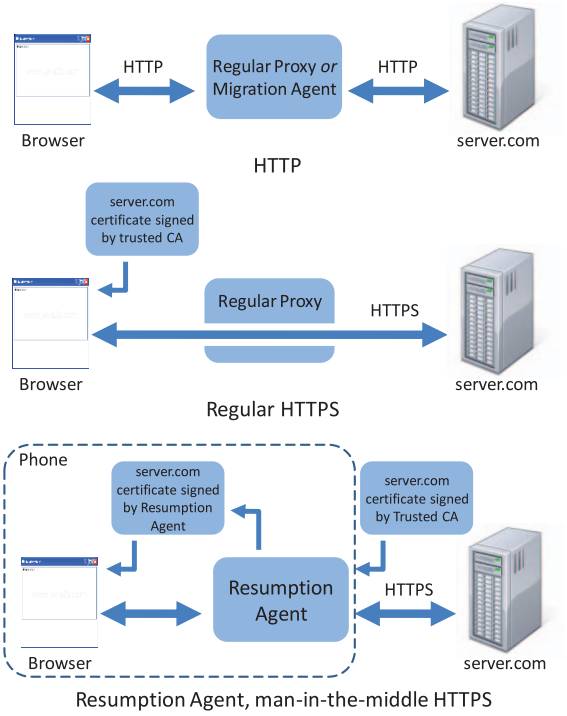

Fig. 8. Regular proxy operation and Resumption Agent man-in-the-middle operation for a browser application.

Resumption Agent employs a novel and secure two-part solution to this challenge. First, it will exploit a *man-in-the-middle attack*. That is, as shown in Fig. 8, Resumption Agent presents itself to the client as the destination server. It then connects to the destination server, and therefore has access to the transferred stream, and can perform the same functionality it does for HTTP. We note that the open source web proxy, *squid*, has built-in support for such man-in-the-middle operation [42].

A standard man-in-the-middle attack by a third party is, however, unable to present the correctly signed certificate to the client application, and depending on system policies, it typically raises a warning to the user. Changing system policies to ignore security certificates would open the door to any man-in-the-middle attack, and is therefore unacceptable. Indeed, to maintain security, the certificate check must be strictly enforced.

The second part of our solution addresses this challenge without compromising security regarding an external man-in-the-middle attacker. All computer systems, including our iPhones, depend on a number of preinstalled Certificate Authorities (CAs) to sign and validate all server certificates. Since Resumption Agent runs fully on the device and is *not* a third party, it can install its own local CA on the device without compromising security. This is possible on the iPhone [43] as well as other platforms, such as Android [44]. Resumption Agent can then sign the certificates it presents to applications, preventing applications from

displaying warning messages. Resumption Agent has to create a new certificate once for each HTTPS domain the user accesses. We have measured the overhead of certificate generation on the iPhone 3GS to be on average 1.7 seconds, with a standard deviation of 1.2 seconds, measured over 100 experiments.

To maintain security, it is imperative to strictly enforce certificate verification between clients and servers. Therefore, Resumption Agent itself verifies the server security certificate instead of the application (e.g., the browser). If a server's certificate is not correctly signed, Resumption Agent (instead of the application) displays a warning to the user. The user can then decide whether to continue or forgo a potentially unsecure connection. We conjecture that a consistent warning for invalid certificates from Resumption Agent may be more understandable to end users, compared to inconsistent application specific warnings. Therefore, by providing a consistent UI, Resumption Agent may possibly reduce bad user decisions.

Finally, to improve protection against rouge rogue applications that may be running directly on the phone, it is important for Resumption Agent to generate a unique CA for every device, to prevent fake certificates copied across devices running Migration Agent.

4.3 Trace-Based Evaluation

In this section, we demonstrate the efficacy of Wait-n-Migrate and Resumption Agent using our field collected traces of real-life usage.

To evaluate Wait-n-Migrate, we calculate the percentage of flows that are successfully transferred between networks for different wait-time values, assuming a time uniform distribution for the system initiating a switch. For example, with a wait-time of 10 seconds, there is a 50 percent chance that a 20 second lifetime flow is migrated successfully. To evaluate Resumption Agent, we measure the feasibility of resuming video streaming and browsing, and show that both YouTube and the majority of the websites participants most commonly visited indeed support resuming.

4.3.1 Wait-n-Migrate

As mentioned in Section 4.1, Wait-n-Migrate requires both networks/interfaces to be connected simultaneously, at least for the duration of the migration. For this evaluation, we assume the device intends to migrate all existing flows to a new network. To assist system designers to set appropriate wait-times in accordance to their application tradeoffs, we use our traces to calculate the percentage of flows that Wait-n-Migrate can successfully migrate to the new network without disruption for different values of wait-time and for different applications, presented in Fig. 9. For example, Wait-n-Migrate successfully migrates all web flows for 90 and 95 percent of cases for wait-time values of 10 and 100 seconds, respectively.

As discussed in Section 4.1, the special case of Wait-n-Migrate that can be employed if a system can only remain connected to one network. The special case waits for the moment where there are no ongoing flows to switch the network. For our evaluation, we assume that this special case of Wait-n-Migrate waits for the moment when there are no web flows, to switch between networks. We have used our traces to calculate the percentage of flows that the special case of Wait-n-Migrate can migrate to the new network without disruption in this manner, shown in Fig. 9. Since the special case does not wait for presumably noninteractive flows (i.e., nonweb) to end, we can see a significantly larger number of disconnections for those flows. The special case of Wait-n-Migrate performs relatively close to Wait-n-Migrate for web flows, as there are rarely multiple web flows in our traces, as shown in Section 3. However, we expect that increased complexity and multitasking in future applications will reduce the performance of the special case of Wait-n-Migrate.

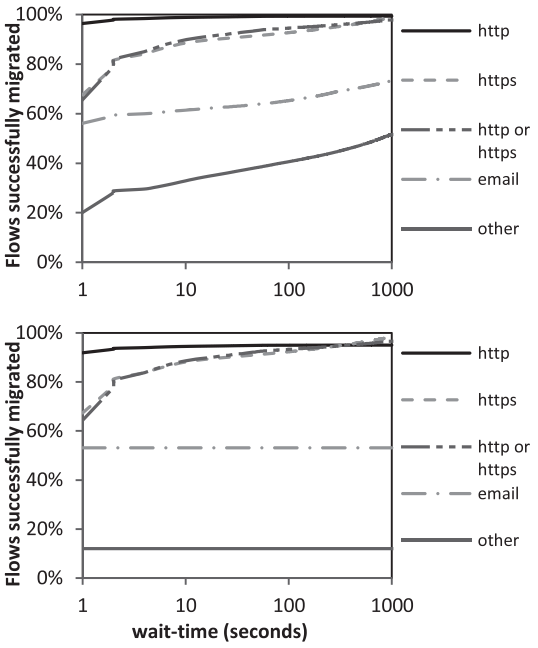

Fig. 9. Performance of Wait-n-Migrate (top) and the special case without simultaneous network connectivity (bottom), according to application and for different wait-times. Performance is measured using our field-collected traces, and presented as the percentage of flows successfully migrated to the target network.

4.3.2 Resumption Agent

We have studied the applicability of Resumption Agent for two important applications, the web browser and the YouTube application. We have tested YouTube and it is fully supported by Resumption Agent; the stream is based on standard HTTP protocols and our experiments show that YouTube servers indeed support resuming videos at an arbitrary location.

We evaluate the applicability of Resumption Agent for web browsing by identifying whether it can be effective for the top 100 websites our users have visited. We used our user study logs to generate the list of top 100 websites our users visit. For each of these 100 sites, we measure the resume capability of the website's homepage and its embedded media (e.g., images). We test the homepages (i.e., top page) because we found that many deeper, pages may depend on previous state information, for example, a specific referrer, cookies, or user login. We crawl these sites both as an iPhone browser and as a desktop browser, set through the *User-Agent* HTTP header. Every crawl, we download each item three times, twice in full, and once

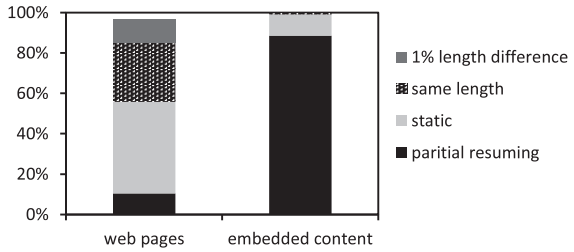

Fig. 10. Most webpages and all of their embedded content (among our top 100 pages) are supported by Resumption Agent. Webpages and/or content that do not support partial resuming need to be restarted from the beginning.

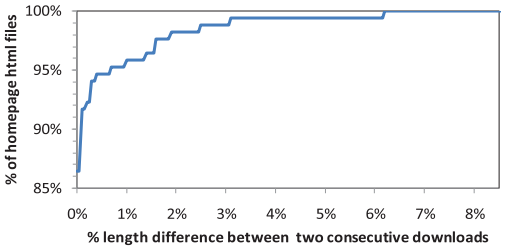

Fig. 11. Nonstatic webpages often have the same or similar content lengths: CDF of the length differences of two consecutive downloads (among the top 100 pages accessed by our users).

from the middle of the transfer to determine 1) if the item supports resuming, and 2) if the item is static. We present the results for the iPhone and desktop browsers together, since they were similar.

As shown in Fig. 10, we found that 100 percent of embedded media is static and, therefore, supported by Resumption Agent. Ninety-one percent of those support resuming from the middle of a transfer; i.e., without the need to retransfer the already transferred part. Among the HTML homepages, 57 percent were static, and 9 percent had the same content length between our two consecutive downloads, but had slightly different content. Furthermore, most others had content lengths very close to each other. Fig. 11 shows the CDF for the length differences between two consecutive downloads for our top 100 pages. We can see that another 30 percent had content lengths within 1 percent of each other. Therefore, we expect them to be supported by Resumption Agent, as described in Section 4.2.1. We note that only 16 percent of the HTML pages can be resumed from the middle of the transfer, versus 89 percent of the embedded content. The remaining pages that do not support HTTP resume functionality incur an extra overhead of redownloading the already transferred part, but can still be resumed transparently to the application.

Finally, while none of the embedded content used HTML tags to disallow caching, we observed that 30 percent of the HTML pages were marked as such. However, of the HTML pages that disallowed caching, 36 percent in fact had static content, and 44 percent had content with the same length. Therefore, we conjecture that the no-cache response header may possibly be ignored by Resumption Agent, even though it is in violation of the expectations of the content provider.

5 iPhone-Based Implementation

We implemented both the Wait-n-Migrate and the Resumption Agent mechanisms on the iPhone 3GS platform and measured their system overhead to be negligible. While the iPhone is a closed platform, a jailbreak has been consistently available, making it possible to develop low-level system software and implement these mechanisms.

We have constrained our solution to support legacy applications. Our design can be extended to every major OS with minimal modification; however, some implementation details are OS specific, as described below.

5.1 Wait-n-Migrate

The implementation of Wait-n-Migrate realizes four functions: monitoring flows, selecting the primary network interface, terminating individual TCP flows, and disabling a network interface:

1. *Flow monitoring*. An intelligent network switching policy requires detailed knowledge of flow properties. For example, it may want to force the migration of high-bandwidth flows with long durations immediately while switching to Wi-Fi. Toward this end, Wait-n-Migrate continuously records flow statistics, such as application, duration, destination, and bandwidth, for all flows. This information is reported in real time, as well as kept in a database which is made available to the switching policy.
2. *Selecting primary network*. The implementation of Wait-n-Migrate depends on the ability to modify the system's routing tables to direct all new flows through the new network. The routing table consists of a set of prioritized rules dictating which interface and gateway to use for establishing new outgoing sockets. Note that existing sockets for prior established flows continue to use their original interface. Furthermore, all common operating systems have a routing table which they allow to be modified through well-documented system calls. While it is possible to directly modify the routing table on the iPhone, we found that any modification to the primary default gateway triggered the system to reset the routing table. Instead, we were able to use the *scutil* command to change the priority of the networks; this in turn automatically changes the routing table appropriately, as well as the DNS settings, and sends a system wide notification of the network change (as it typically does when switching interfaces). The overhead for invoking a switch is small, as it simply changes a system setting, and takes less than 300 ms to complete. scutil is an OSX specific tool, though other OSs provide proprietary methods to select the primary network interface. Conveniently, the iPhone does not disable the cellular interface while Wi-Fi is connected, as some platforms, such as Android, do. This, however, does not have a power impact as the phone must leave the cellular interface on to receive calls.
3. *Terminating flows*. As mentioned previously, it may be necessary to force the migration of specific flows, such as those that are known to be of long

```
Establish Connection to Server
Forward Client Headers
Forward Server's Reply Headers to Client
While download_unfinished && tries < MAXRETRIES {
    Forward CHUNKSIZE bytes of data
    If disconnect_signaled
        Disconnect connection to server
    If disconnected && eligible for resume
        Reconnect to server
        Forward original headers
            If server supports ranges
                Request range starting at prior to disconnect
            else
                Download and discard previously downloaded data
    Tries++ }
Close sockets
```

Fig. 12. Pseudocode for Resumption Agent.

duration. We have achieved this by porting *tcpkill* to the iPhone platform. tcpkill has been ported to all major kernels, including Darwin, Windows, FreeBSD, OpenBSD, HP-UX, AIX, Solaris, and Linux. tcpkill uses *libpcap* and *libnet* to detect/monitor the TCP stream and inject a TCP RST packet that kills the connection. When the application reconnects, it is automatically routed through the new network.

4. *Disabling network*. Wait-n-Migrate provides a mechanism to disable the entire network being migrated from. This can be useful after individual flows have been appropriately dealt with, or none of the flows require special treatment, depending on policy. Every major OS has methods to disable network interfaces. For UNIX-based OSs this is typically "*ifconfig* interface down." Unfortunately, this method currently does not work on the iPhone; however, similar functionality can be achieved through the *scutil* or *ipconfig* commands. Additionally, low-level *ioctl* calls can also accomplish this behavior.

5.2 Resumption Agent

We implemented Resumption Agent in 1,400 lines of C code; it can be built and run on any POSIX compliant system, including Linux and iOS. Resumption agent is similar to other proxies, such as squid, in that it acts as a relay point for Internet communication between clients and servers, complies with HTTP 1.0/1.1 specifications, and handles multiple concurrent connections.

Our implementation (Fig. 12) leverages standard UNIX sockets and multithreading. When Resumption Agent starts it initializes a pool of worker threads, using *libpthread*, to handle concurrent requests. Each thread handles one request at a time; however, more threads can be dynamically created to handle heavy loads. Creating the threads in advance reduces latency for handling incoming requests. Next, the agent uses the *listen*() command to start listening on a predefined TCP port, such as 8080, for incoming connections.

When a new incoming client connection is received, Resumption Agent uses the connection's file descriptor to hand the request to a worker thread. The worker thread then uses asynchronous nonblocking UNIX IO, *read_nio*(), to read from the connection. It parses the client headers, then establishes a connection to that server and forwards the client's request headers to the server. When the server begins answering the request, the worker thread forwards the data back to the client. We note that for each transfer, Resumption Agent only has to process the header data; everything else is simply forwarded without processing overhead, minimizing any performance impact. Furthermore, to reduce CPU usage and bandwidth without increasing latency, the worker threads employ a large read size of 2 KB, or the amount of data available in the system queue, from the server before forwarding it to the client.

The *socket*() implementation usually allows a socket time-out option to be specified, to report a disconnection if no data have been sent after the specified time-out. Unfortunately, while the iPhone appears to implement this option, it failed to function. Thus, we implemented our own time-out detection, with the same behavior. If a time-out is detected, or the socket throws any error, the worker thread reestablishes a connection to the server and attempts to resume the transfer where it left off. The worker will retry up to a predefined number of times, by default 50, before giving up; this keeps the worker from running infinitely and flooding the network, if the server or network becomes unavailable for extended periods of time.

We have measured the performance impact of Resumption Agent to be minimal in normal usage. In particular, Resumption Agent consumes less than 300 KB of memory, and it increases linearly with the number of concurrent transfers. Its mean CPU consumption is negligible when idle, and 3-4 percent when actively handling transfers. Most importantly, we have measured the additional latency introduced by the Resumption Agent to be statistically insignificant over 200 test runs.

6 Example Policy: AutoSwitch

To evaluate the combined effectiveness of the Wait-n-Migrate and Resumption Agent mechanisms, we have developed AutoSwitch, an automatic network interface switching policy. AutoSwitch attempts to offload data from cellular to Wi-Fi as much as possible, with minimum disruptions to the user. AutoSwitch solves a common complaint about Wi-Fi [45]—that it is unreliable or unusable at low signal levels, such as while the user is moving in and out of coverage areas. To achieve this goal, AutoSwitch intelligently switches between wireless networks using Wait-n-Migrate and Resumption Agent, before losing connectivity (e.g., due to mobility). We note that other solutions have been proposed to offload cellular traffic on Wi-Fi, for example, [6], but they typically rely on a mobility gateway/proxy to handle network switches, with inherent latency and deployability drawbacks as discussed in Section 2.2.

6.1 AutoSwitch Design

AutoSwitch attempts to migrate TCP flows from Wi-Fi to cellular before Wi-Fi coverage is dropped or becomes unreliable, and migrate back to Wi-Fi when a reliable Wi-Fi

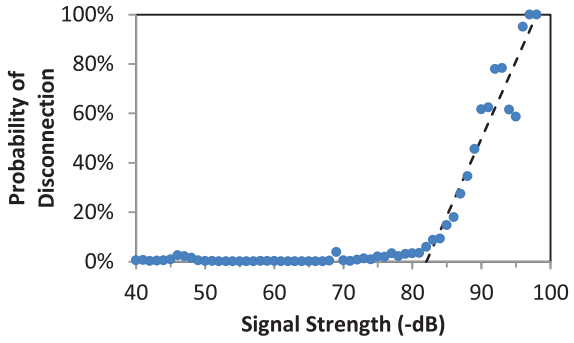

Fig. 13. Probability of disconnection versus Wi-Fi signal strengths.

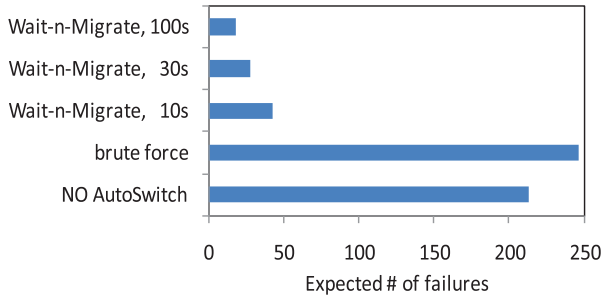

Fig. 14. AutoSwitch significantly reduces the expected number of disruptions in 1,120 hours of interactive usage traces with Wi-Fi enabled, for different wait-time values.

connection becomes available again. For simplicity, we assume that cellular coverage is always available.[2]

Often, in particular for the case of mobility, switching between networks occurs due to forced disconnections. For example, a phone may switch from 3G to a Wi-Fi network when Wi-Fi becomes available, but move out of Wi-Fi coverage shortly afterwards; thus, the phone is forced to switch back to 3G. In such a case, it is too late to effectively use Wait-n-Migrate. However, previous work shows that it is indeed possible to accurately predict network conditions, and therefore initiate the network switch before losing coverage. For example, Breadcrumbs [46] and our previous work [47] predict network conditions for the near and far future, respectively. As our main focus is on flow migration and not on the switching policy, we use a simple yet effective predictor, signal strength [9], [48], to initiate a network switch before losing Wi-Fi coverage.

To determine the policy for switching to and from Wi-Fi, we extended LiveLab for three iPhone 3GS users for three weeks to continuously test and record network disconnections, measured by the ping tool. These three users acted as a sampling tool to measure Wi-Fi reliability at different signal strengths, collecting over 1 million connectivity tests, shown in Fig. 13. We define a Wi-Fi connection as disconnected if all ping tests over a period of 5 seconds are lost, regardless of the reported signal strength. We can see that Wi-Fi starts to become unreliable starting at approximately -82 dBm on iPhone 3GS.

Based on these results, we employ a simple hysteresis over both time and signal strength to reduce erroneous switching. AutoSwitch, using Wait-n-Migrate and Resumption Agent, switches to cellular when a Wi-Fi signal level of -75 dBm or less is maintained over 3 seconds, and switches back to Wi-Fi when Wi-Fi signal strength reaches -70 dBm. We chose the -75 dBm threshold, instead of -82 dBm, to provide a safety zone so that dynamics and fluctuations in the Wi-Fi signal does not make it fall below -82 dBm and cause disruptions. Clearly, a smaller safety zone would reduce the success rate, but transfer a larger portion of data over Wi-Fi.

6.2 Trace-Based Evaluation

We have used the traces from LiveLab to evaluate the efficacy of AutoSwitch using Wait-n-Migrate, during routine interactive usage. As mentioned in Section 6.1, LiveLab provides us with continuous signal strength measurements, but not connectivity measurements. We utilize the Wi-Fi signal strength measurements and the probability of disconnection at different signal levels, presented in Fig. 13, to calculate the expected number of disruptions in a web application. We further assume that if Wi-Fi is not disconnected at a specific signal level in a particular usage session, it will not be disconnected at that signal level for the entire session.

In Fig. 14, we present the expected number of disconnections for AutoSwitch using the Wi-Fi signal strength thresholds identified in Section 6.2. The number of disconnections is calculated for Wait-n-Migrate with wait-times of 10, 30, and 100 seconds, as well as for the case where Wi-Fi is left on (no AutoSwitch), and for the case where AutoSwitch switches between networks in a brute-force manner, where one network is simply disabled and another enabled.

Using 1,120 hours of interactive usage traces with Wi-Fi enabled, for web usage, the users were expected to experience 213 disruptions without AutoSwitch. Employing AutoSwitch using Wait-n-Migrate and a constant wait-time of 10, 30, and 100 seconds, users were expected to experience 80, 87, and 91 percent fewer disconnections, respectively (Fig. 14). In contrast, AutoSwitch with brute-force switching, i.e., without Wait-n-Migrate, slightly increases disconnections to 246, due to false positives.

We must note that users indeed take note of the mobility and coverage limitations of Wi-Fi, as confirmed by our motivational user study from Section 2.1 and prior work [45]. Therefore, they may change their behavior and turn off Wi-Fi or avoid using the phones altogether in conditions they know it is prone to failing. Hence, we expect that the results in this section, obtained from the traces only when Wi-Fi was enabled and users were browsing the web, underestimate the potential benefit from AutoSwitch using Wait-n-Migrate.

6.3 Field Evaluation

We further evaluate AutoSwitch using both Wait-n-Migrate and Resumption Agent on the iPhone platform. For performance evaluation we wrote a script to automatically download a predetermined file over HTTP, from a server that supports resuming, every five seconds. We tested AutoSwitch using transfer sizes of 10 KB, 100 KB, and 1 MB, as well as Wait-n-Migrate alone. We then measured the

2. This was the case for the field measurement of AutoSwitch. Of course, it is possible for AutoSwitch to check for cellular data coverage, and avoid switching from Wi-Fi when there is no cellular data coverage (i.e., bad Wi-Fi is better than no cellular).

Fig. 15. Map of paths travelled in Rice University, for walking (1 km, Blue loop) and driving (3 km, Red loop) scenarios.

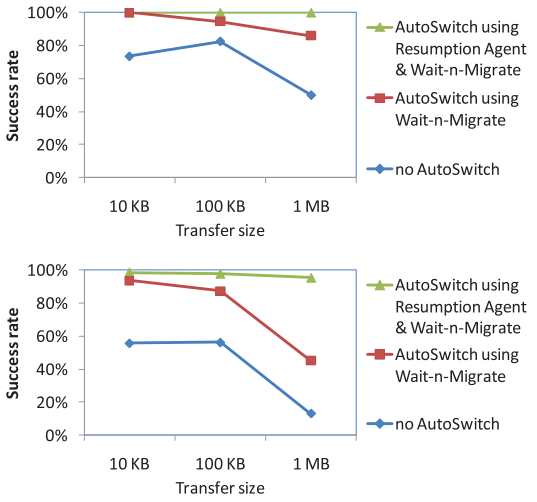

Fig. 16. AutoSwitch significantly increases the success rate of 10-KB, 100-KB, and 1-MB transfers when walking (top) and driving (bottom) on Rice Campus.

number of transfers that were fully completed without errors over two predetermined paths in Rice University, shown in Fig. 15; 1) while walking commonly used paths, and 2) while in a car traveling at approximately 30 km/h along campus roads. The walking path was approximately 1-km long, included indoor areas in two buildings, crossed distinct areas with good to excellent Wi-Fi connectivity (-70 dBm or better signal strength), and the phone was associated with a Wi-Fi access point for 95 percent of the time. The driving path was approximately 3-km long and only had one area of good Wi-Fi signal strength, but the phone was associated with a Wi-Fi access point for 80 percent of the time. Each test run lasted approximately 1 hour, and included approximately 350 attempts for each transfer size.

The success rates of transfers, as observed by our script, are shown in Fig. 16. As expected, due to Wi-Fi signal variations, there are a significant number of failed transfers without AutoSwitch. Using AutoSwitch in conjunction with Wait-n-Migrate significantly reduced the number of disruptions. Furthermore, since the server supported resuming, Resumption Agent, used in conjunction with Wait-n-Migrate, was able to further reduce disruptions, completely eliminating them while walking, and increased the success rate while driving to over 95-99 percent for different file sizes.

7 Discussion

Our work focuses on providing system mechanisms for migrating flows between networks. Various policies have been proposed to switch between or aggregate networks. AutoSwitch is one such policy, and unambiguously demonstrates the effectiveness of Wait-n-Migrate and Resumption Agent in supporting seamless flow migration. These system mechanisms can be utilized to enable the immediate deployment of many other performance and efficiency-enhancing policies studied in literature, without practical deployment issues:

Multihoming/Load Balancing. When used for load balancing and multihoming, Resumption Agent has the key advantage of knowing the length of a flow at its very early stages, through the HTTP response headers, as well as the properties and conditions of the available networks. This allows Resumption Agent to intelligently allocate each flow on the appropriate network interface.

Striping. Resumption Agent can be extended to support striping larger transfers, as long as the content supports resuming, i.e., download different parts of a transfer simultaneously through different network interfaces, and then amalgamate these chunks before sending them to the client. For striping content that contains dynamic parts, as described in Section 4.2.1, it is necessary to ensure the dynamic portions are downloaded in single chunks.

Mirroring. For transfers that do not support striping, or that are very small compared to the connection latency, Resumption Agent can be extended to simultaneously request the transfer on multiple networks, and return whichever finishes first. While this reduces efficiency, it can reduce user perceived latency, especially under highly varying network environments.

Preemptive network switching. When Resumption Agent is aware of an impending network switch, it can establish a connection over the new network and request the remaining portion of the flow, *before killing the existing flow*. This allows the Resumption Agent to further minimize the latency incurred when resuming a flow.

8 Conclusion

We presented a first-of-its-kind characterization of IP traffic on modern smartphones using traces collected in real-life usage of 27 iPhone 3GS users over a period of three months. We show that the traffic is almost exclusively TCP, and TCP flows are often short lived and rarely concurrent for interactive applications.

Driven by these findings, we devised two novel and complementary system mechanisms to migrate TCP flows between networks without network or application support: Wait-n-Migrate and Resumption Agent. While Wait-n-Migrate significantly decreases, or even eliminates connectivity gaps when switching between networks, Resumption Agent opportunistically resumes flows across connectivity disruptions and network switches. Combined,

these two system mechanisms mitigate, and in many cases eliminate, the impact of widely varying network conditions on mobile applications, as we demonstrate using our implementation, AutoSwitch. The seamless flow migration without network support collectively enabled by Wait-n-Migrate and Resumption Agent allows for immediate deployment of performance and efficiency-enhancing policies, including multihoming and traffic offloading.

REFERENCES

[1] A. Rahmati and L. Zhong, "Context-for-Wireless: Context-Sensitive Energy-Efficient Wireless Data Transfer," *Proc. ACM Int'l Conf. Mobile Systems, Applications and Services (MobiSys),* pp. 165-178, 2007.

[2] W. Qadeer, T.S. Rosing, J. Ankcorn, V. Krishnan, and G. De Micheli, "Heterogeneous Wireless Network Management," *Proc. Workshop Power Aware Computer Systems (PACS),* pp. 86-100, 2003.

[3] N. Thompson, G. He, and H. Luo, "Flow Scheduling for End-Host Multihoming," *Proc. IEEE INFOCOM,* 2006.

[4] S. Kandula, K.C.-J. Lin, T. Badirkhanli, and D. Katabi, "FatVAP: Aggregating AP Backhaul Capacity to Maximize Throughput," *Proc. USENIX Symp. Networked Systems Design and Implementation (NSDI),* 2008.

[5] P. Rodriguez, R. Chakravorty, J. Chesterfield, I. Pratt, and S. Banerjee, "MAR: A Commuter Router Infrastructure for the Mobile Internet," *Proc. ACM MobiSys* 2004.

[6] A. Balasubramanian, R. Mahajan, and A. Venkataramani, "Augmenting Mobile 3G Using WiFi," *Proc. ACM MobiSys,* 2010.

[7] B. Han, P. Hui, V. Kumar, M. Marathe, G. Pei, and A. Srinivasan, "Cellular Traffic Offloading through Opportunistic Communications: A Case Study," *Proc. ACM Int'l Workshop Challenged Networks (CHANTS),* 2010.

[8] T. Pering, Y. Agarwal, R. Gupta, and R. Want, "CoolSpots: Reducing the Power Consumption of Wireless Mobile Devices with Multiple Radio Interfaces," *Proc. ACM MobiSys,* pp. 220-232, 2006.

[9] A. Giannoulis, M. Fiore, and E.W. Knightly, "Supporting Vehicular Mobility in Urban Multi-Hop Wireless Networks," *Proc. ACM MobiSys,* 2008.

[10] A. Lai and J. Nieh, "Limits of Wide-Area Thin-Client Computing," *ACM SIGMETRICS Performance Evaluation Rev.,* vol. 30, no. 1, pp. 228-239, 2002.

[11] J. Huang, Q. Xu, B. Tiwana, Z. Mao, M. Zhang, and P. Bahl, "Anatomizing Application Performance Differences on Smartphones," *Proc. ACM MobiSys,* pp. 165-178, 2010.

[12] A. Agresti and B.A. Coull, "Approximate Is Better than 'Exact' for Interval Estimation of Binomial Proportions," *The Am. Statistician,* vol. 52, no. 2. pp. 119-126, 1998.

[13] E. Gustafsson and A. Jonsson, "Always Best Connected," *IEEE Wireless Comm.,* vol. 10, no. 1, pp. 49-55, Feb. 2003.

[14] K. Pahlavan, P. Krishnamurthy, A. Hatami, M. Ylianttila, J. Makela, R. Pichna, and J. Vallstron, "Handoff in Hybrid Mobile Data Networks," *IEEE Personal Comm.,* vol. 7, no. 2, pp. 34-47, Apr. 2000.

[15] D. Maltz and P. Bhagwat, "MSOCKS: An Architecture for Transport Layer Mobility," *Proc. IEEE INFOCOM,* 1998.

[16] R. Chalmers and K. Almeroth, "A Mobility Gateway for Small Device Networks," *Proc. IEEE Second Int'l Conf. Pervasive Computing and Comm. (PerCom),* 2004.

[17] P. Sharma, S. Lee, J. Brassil, and K. Shin, "Handheld Routers: Intelligent Bandwidth Aggregation for Mobile Collaborative Communities," *Proc. IEEE First Int'l Conf. Broadband Networks (BroadNets),* 2004.

[18] H. Pucha and Y. Hu, "Overlay TCP: Ending End-to-End Transport for Higher Throughput," *Proc. ACM SIGCOMM Poster,* 2005.

[19] H. Hsieh and R. Sivakumar, "pTCP: An End-to-End Transport Layer Protocol for Striped Connections," *Proc. IEEE Int'l Conf. Network Protocols (ICNP),* 2002.

[20] H. Han, S. Shakkottai, C. Hollot, R. Srikant, and D. Towsley, "Overlay TCP for Multi-Path Routing and Congestion Control," *IEEE/ACM Trans. Networking,* 2006.

[21] S. Kandula, D. Katabi, S. Sinha, and A. Berger, "Dynamic Load Balancing without Packet Reordering," *ACM SIGCOMM Computer Comm. Rev.,* vol. 37, no. 2, pp. 51-62, 2007.

[22] C. Traw and J. Smith, "Striping within the Network Subsystem," *IEEE Network,* vol. 9, no. 4, pp. 22-32, July/Aug. 1995.

[23] H. Sivakumar, S. Bailey, and R.L. Grossman, "PSockets: The Case for Application-Level Network Striping for Data Intensive Applications Using High Speed Wide Area Networks," *Proc. ACM/IEEE Conf. Supercomputing,* 2000.

[24] H. Hsieh, K. Kim, Y. Zhu, and R.A. Sivakumar, "Receiver-Centric Transport Protocol for Mobile Hosts with Heterogeneous Wireless Interfaces," *Proc. ACM MobiCom,* 2003.

[25] S. Kim and J. Copeland, "TCP for Seamless Vertical Handoff in Hybrid Mobile Data Networks," *Proc. IEEE Global Comm. Conf. (GlobeCom),* 2003.

[26] K.-H. Kim, Y. Zhu, R. Sivakumar, and H.-Y. Hsieh, "A Receiver-Centric Transport Protocol for Mobile Hosts with Heterogeneous Wireless Interfaces," *Wireless Networking* vol. 11, no. 4, pp. 363-382, 2005.

[27] M. Stemm and R. Katz, "Vertical Handoffs in Wireless Overlay Networks," *Mobile Networks and Applications,* vol. 3, no. 4, pp. 335-350, 1998.

[28] C. Perkins, S. Alpert, and B. Woolf, *Mobile IP: Design Principles and Practices.* Addison-Wesley Longman, 1997.

[29] C. Perkins, "Mobile IP," *IEEE Comm. Magazine,* vol. 35, no. 5. pp. 84-99, May 1997.

[30] S. Nirjon, A. Nicoara, C.-H. Hsu, J.P. Singh, and J. Stankovic, "MultiNets: Policy Oriented Real-Time Switching of Wireless Interfaces on Mobile Devices," *Proc. IEEE Real-Time and Embedded Technology and Applications Symp. (RTAS),* 2012.

[31] A.C. Snoeren, D.G. Andersen, and H. Balakrishnan, "Fine-Grained Failover Using Connection Migration," *Proc. USENIX Symp. Internet Technologies and Systems,* 2001.

[32] T. Alperovich and B. Noble, "The Case for Elastic Access," *Proc. ACM Int'l Workshop Mobility in the Evolving Internet Architecture (MobiArch),* 2010.

[33] R. Chandra and B. Bahl, "MultiNet: Connecting to Multiple IEEE 802.11 Networks Using a Single Wireless Card," *Proc. IEEE INFOCOM,* 2004.

[34] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan, and D. Estrin, "Diversity in Smartphone Usage," *Proc. ACM MobiSys,* 2010.

[35] H. Falaki, D. Lymberopoulos, R. Mahajan, S. Kandula, and D.A. Estrin, "First Look at Traffic on Smartphones," *Proc. Internet Measurement Conf. (IMC),* 2010.

[36] A. Rahmati, C. Shepard, A. Nicoara, L. Zhong, and J.P. Singh, "MobiCom 2010 Poster: Mobile TCP Usage Characteristics and the Feasibility of Network Migration without Infrastructure Support," *ACM SIGMOBILE Mobile Computer Comm. Rev.,* vol. 14, pp. 10-12, 2010.

[37] A. Rahmati, C. Shepard, C. Tossell, A. Nicoara, L. Zhong, P. Kortum, and J. Singh, "Seamless Flow Migration on Smartphones without Network Support," Technical Report 2010-1214, Rice Univ., http://arxiv.org/abs/1012.3071, 2010.

[38] C. Shepard, A. Rahmati, C. Tossell, L. Zhong, and P. Kortum, "LiveLab: Measuring Wireless Networks and Smartphone Users in the Field," *Proc. Workshop Hot Topics in Measurement & Modeling of Computer Systems (HotMetrics),* 2010.

[39] AdMob, "Oct. 2009 Mobile Metrics Report," http://metrics.admob.com/2009/11/oct.-2009-mobile-metrics-report, Oct. 2009.

[40] F. Dogar and P. Steenkiste, "Catnap: Exploiting High Bandwidth Wireless Interfaces to Save Energy for Mobile Devices," *Proc. ACM MobiSys,* 2010.

[41] M. Buchanan, "Gizmodo Blog: How Multitasking Works on a Phone," http://gizmodo.com/5527407/giz-explains-how-multitasking-works-on-a-phone, Apr. 2010.

[42] "Squid-in-the-Middle SSL Bump," *Squid-Cache Wiki,* http://wiki.squid-cache.org/Features/SslBump, 2013.

[43] "iPhone Certificate Flaws," *Cryptopath Blog,* http://cryptopath.wordpress.com/2010/01/29/iphone-certificate-flaws, 2013.

[44] "Adding .cer Certificates on Your Android Phone," *It's All About Everything Blog,* http://www.abtevrythng.com/2010/06/adding-cer-certificates-on-your-android.html, 2013.

[45] A. Rahmati and L. Zhong, "A Longitudinal Study of Non-Voice Mobile Phone Usage by Teens from an Underserved Urban Community," Technical Report 0515-09, Rice Univ., 2009.

[46] A.J. Nicholson and B.D. Noble, "BreadCrumbs: Forecasting Mobile Connectivity," *Proc. ACM MobiCom,* pp. 46-57, 2008.

[47] A. Rahmati and L. Zhong, "Context-Based Network Estimation for Energy-Efficient Ubiquitous Wireless Connectivity," *IEEE Trans. Mobile Computing*, vol. 10, no. 1, pp. 54-66, Jan. 2011.
[48] S.K. Kim, C.G. Kang, and K.S. Kim, "An Adaptive Handover Decision Algorithm Based on the Estimating Mobility from Signal Strength Measurements," *Proc. Vehicular Technology Conf. (VCT)*, 2004.

Ahmad Rahmati received the BS degree in computer engineering from the Sharif University of Technology in 2004 and the MS and PhD degrees from the Department of Electrical and Computer Engineering at Rice University in 2008 and 2012. He is a senior staff scientist at the Mobile and Wireless Group at Broadcom. His publications received the ACM MobileHCI Best Paper Award in 2007 and have been featured twice as the spotlight paper of the *IEEE Transactions on Mobile Computing*, in 2010 and 2011. His research interests include mobile and wireless system design and applications, context-aware computing through sensing and statistical learning on large data, as well as human factors and HCI. He is a member of the IEEE.

Clayton Shepard received the BS degree in 2008 and the MS degree in 2012 from Rice University, where he is currently working toward the PhD degree. He is a member of the Rice Efficient Computing Group, led by Dr. Lin Zhong. In 2008, he was a visiting researcher with Motorola's Advanced Research and Technology lab, where he also interned in 2007. His research interests include mobile systems and ubiquitous low-power computing. His current research focus is many-antenna base stations. In 2011 and 2012, he interned with Bell Labs, Alcatel-Lucent. He received the NDSEG Fellowship Award in 2011. He is a student member of the IEEE.

Chad C. Tossell received the BS degree in psychology from the University of California, Berkeley in 2003, the MS degree in applied psychology from Arizona State University in 2006, and the PhD degree in psychology from Rice University in 2012. He currently leads training research for the US Air Force (USAF) Research Laboratory at Wright-Patterson Air Force Base, Ohio. He oversees research and development efforts aimed at enhancing human and team performance within command and control, intelligence, and cyberspace domains. His other research interests include the use and usability of mobile systems, personalization of technology, and the human factors of voting with handheld devices.

Lin Zhong received the BS and MS degrees from Tsinghua University in 1998 and 2000, respectively, and the PhD degree from Princeton University in September 2005. He is an associate professor in the Department of Electrical & Computer Engineering, Rice University. He received the US National Science Foundation CAREER award and Best Paper Awards from ACM MobiSys 2011, IEEE PerCom 2009, and ACM MobileHCI 2007. A paper he coauthored was identified as one of the 30 most influential papers in the first 10 years of the Design, Automation & Test in Europe conference. His research interests include mobile computing, human-computer interaction, and nanoelectronics. He is a member of the IEEE.

Philip Kortum received the PhD degree from the University of Texas at Austin in 1994. He is a faculty member in the Department of Psychology at Rice University in Houston, Texas. Prior to joining Rice, he was at SBC Laboratories (now AT&T Laboratories) for almost a decade doing human factors research and development in all areas of telecommunications. He continues to work on the research and development of user-centric systems in both the visual (web design, equipment design, image compression) and auditory domains (telephony operations, mobile computing and interactive voice response systems).

Angela Nicoara received the computer science degree from the Politehnica University of Timisoara in 2002 and the PhD degree from ETH Zurich in 2007. She has been a senior research scientist at Deutsche Telekom Innovation Labs, Silicon Valley Innovation Center, since 2008. Her research interests encompass services and mobile platforms, adaptive software architectures, middleware, virtual machines, and distributed systems. Her current research activities include the development of open and programmable mobile platforms (e.g., Android) and novel IT services to shape the emerging trends in fixed and mobile infrastructure and services sectors. She is a member of the IEEE.

Jatinder Singh received the BS degree in electrical engineering from the Indian Institute of Technology, Delhi, in 2005, where he graduated at the top of his class with an Institute Silver Medal. He received the MS and PhD degrees in electrical engineering from Stanford University, where he received the Stanford Graduate Fellowship and Deutsche Telekom Fellowship. He is the director of Mobile Innovation Strategy at the Palo Alto Research Center (PARC) and a consulting associate professor in the Department of Electrical Engineering at Stanford University. He was previously the vice president of research with Deutsche Telekom, the parent company of T-Mobile. He is a member of the IEEE.